68' MICRO JOURNAL

Australia A $4.75 | New Zealand NZ $ 6.50
Singapore S $9.45 | Hong Kong H $23.50
Malaysia M $9.45 | Sweden 30:-SEK

$2.95USA

OS-9 Atari Amiga Mac S-50

6800 6809 68008 68000 68010 68020 68030

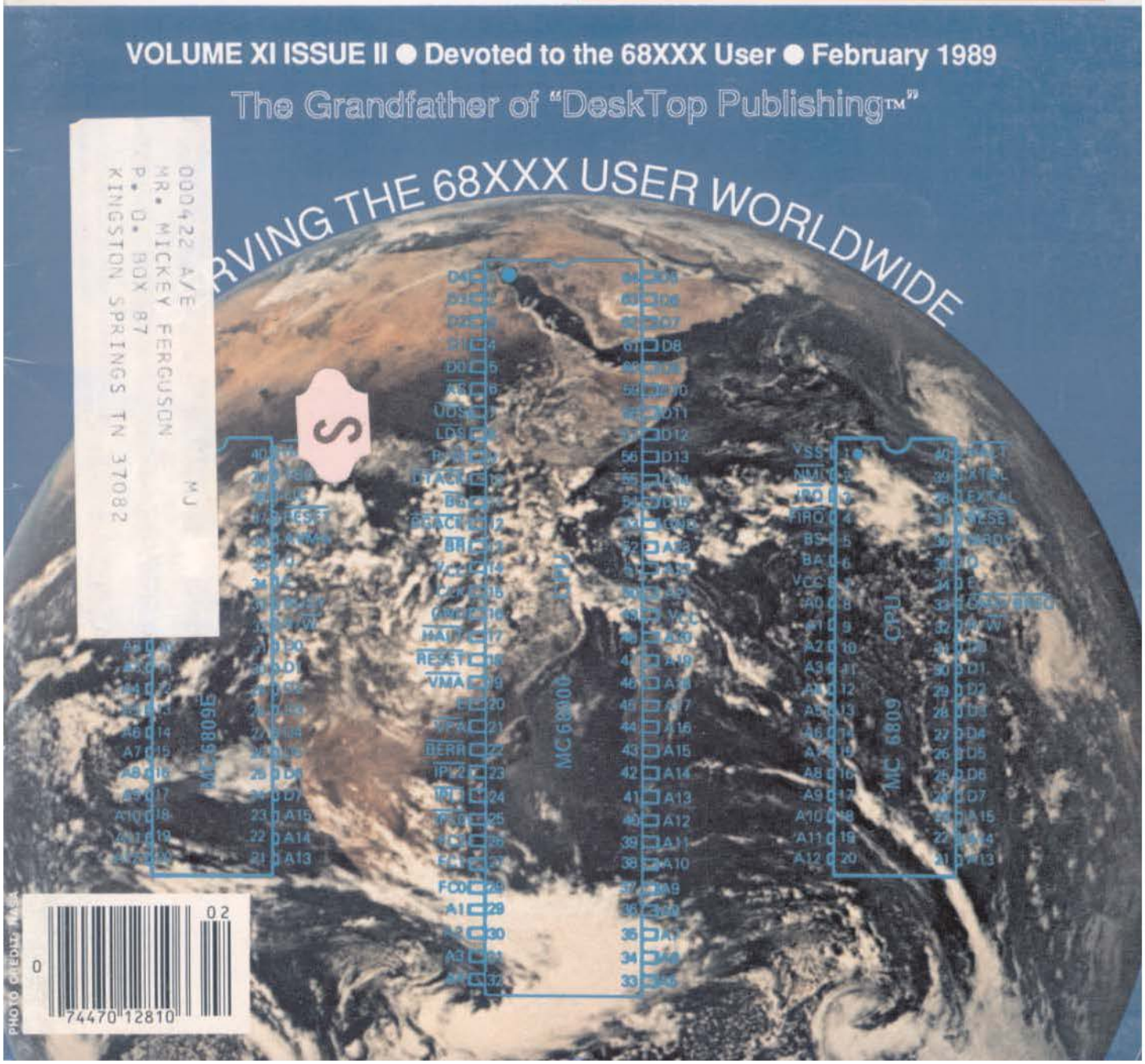
The Magazine for Motorola CPU Devices For Over a Decade!

This Issue:

OS-9 SK*DOS Atari Amiga FLEX Macintosh *A User Contributor Journal* And Lots More!

VOLUME XI ISSUE II ● Devoted to the 68XXX User ● February 1989

The Grandfather of "DeskTop Publishing™"



0 74470 12810 02

PHOTO CREDIT: NASA

WHO DO YOU CALL WHEN YOUR DEBUGGER WON'T DEBUG?

The problem with most real-time operating systems is simple, they're not an integrated solution. You end up dealing with a multitude of suppliers for languages, compilers, debuggers and other important development tools. And when something does go wrong, it can be a frustrating experience trying to straighten out the mess.

Why Not Try the Microware One-Stop Total Solution?

Microware's OS-9 Real-Time Operating System is a total integrated software system, not just a kernel. We offer an extensive set of development tools, languages, I/O and Kernel options. ***And this total integrated solution is entirely designed, built and supported by the same expert Microware team.***

Modularity Lets YOU Choose Just What You Need.

The modular design of OS-9 allows our Operating System to adapt as your requirements change. OS-9 can support a complete spectrum of applications — from embedded ROM-based code in board-level products all the way up to large-scale systems.

The OS-9 Success Kit

A Total Integrated Solution for Your Next Project

Development Tools:
- C Source Level Debugger
- Symbolic Debugger
- System State Debugger
- uMACS Text Editor
- Electronic Mail
- Communications
- Super Shell

Kernel Options:
- MMU (Security Protection) Support
- Math Coprocessor Support

Languages:
- C*
- Basic
- Pascal
- Fortran
- Ada**
- Assembler*

I/O Options:
- SCSI, SASI & SMD Disks
- 3-, 5-, 8-inch Diskettes
- Magnetic Tape
- Ethernet - TCP/IP
- Arcnet - OS-9/Net

* Assembler or UNIX versions available
** VAX hosted

Support is Part of the Package.

Microware is proudly setting the industry's standard for customer support. You'll find professional and comprehensive technical documentation and a Customer Hotline staffed by courteous and authoritative software engineers.

So stop messing with simple kernels and independent suppliers. Call Microware today and find out more about the "One-Stop Integrated Solution" with OS-9!

microware® OS-9

Microware Systems Corporation
1900 N.W. 114th Street
Des Moines, Iowa 50322
Phone: 515/224-1929

Western Regional Office
4401 Great America Parkway
Santa Clara, California 95054
Phone: 408/980-0201

Microware Japan Ltd.
41-19 Honcho 4-Chome
Funabashi City
Chiba 273, Japan
Phone: 0474 (22) 1747

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	3.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
        {
                register long i;
                for (i=0; i < 999999; ++i);
        }
```

Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS,
Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
buffered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

OS-9

OS-9 Professional Ver *Includes C Compiler	$850.00
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Oregon Software Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Scalpel+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Scalpel+ (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.

Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below RS232 Adapter	165.00

Each card supports 4 additional ser. ports (total of 36 serial ports supported)

60 line Parallel I/O card	398.00

Uses 3 68230 Interface/Timer chips, 6 groups of 8 lines each, separate buffer direction control for each group.

Prototype Board	75.00

area for both dip and PGA devices & a pre-wired memory area up to 512K DRAM.

SBC-AN	475.00

interface between the system and ARCNET modified token-passing LAN, fiber optics optional - call. LAN software drivers 120.00

Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Scalpel+ $695.00. SAVE $300.00
Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quotes. Discounts apply after the sale as well.

The **PRO!**

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!

This is *HEAVY DUTY* Country!

For a limited time we will offer a $400 trade-in on your old 68XXX SBC. Must be working properly and complete with all software, cables and documentation. Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

NEW LOWER PRICES

25 Mbyte HD ~~$4299.80~~ $3749.80

85 Mbyte HD ~~$5248.80~~ $4548.80

Note: Only Professional OS-9 Now Available (68020 Version) Includes ($500) C Compiler - 68020 & 68881 Supported - For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division

A Decade of Quality Service
Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Cheryl Hodge

Contributing & Associate Editors

Ron Anderson — Dr. E.M. "Bud" Pass
Ron Voigts — Art Weller
Doug Lurie — Dr. Theo Elbert
Ed Law — & Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.
"Over a Decade of Service"
"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343
Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal (ISSN 0194-5025) is published 12 times a year by Computer Publishing Inc. Second Class Postage paid at Hixson, TN. and additional entries. POSTMASTER: send address changes to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please . do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

PAT - JUST

PAT

With 'C' Source

$229.00

All OS-9 68XXX Systems

PAT FROM S. E. MEDIA -- A FULL FEATURED SCREEN ORIENTED TEXT EDITOR with all the best of PIE. For those who swore by and loved **PIE**, this is for **YOU! All PIE features & much more!** Too many features to list. And if you don't like ours, change or add your own. **C source included.** Easily configured to your CRT terminal, with special configuration section. No sweat!

68008 - 68000 - 68010 - 68020 OS-9 68K $229.00

COMBO
PAT/JUST

Special $249.00

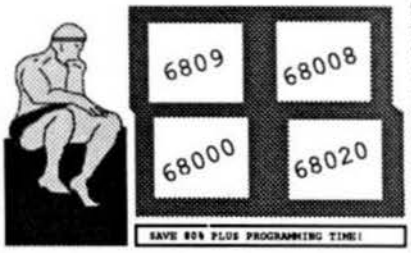
JUST

JUST from S. E. MEDIA - - Text formatter written by Ron Anderson; for dot matrix printers, provides many unique features. Output formatted to the display. User configurable for adapting to other printers. Comes set-up for Epson MX80 with Graflex. Up to 10 imbedded printer control commands. Compensates for double width printing. Includes normal line width, page numbering, margin, indent, paragraph, space, vertical skip lines, page length, centering, fill, justification, etc. Use with **PAT** or any other text editor. The ONLY stand alone text processor for the 68XXX OS-9 68K, that we have seen. And at a very **LOW PRICE!** Order from: S.E. MEDIA - see catalog this issue.

68008 - 68000 - 68010 - 68020 OS-9 68K
With 'C' source $79.95

CLOSE OUT SPECIAL

SCULPTOR



From the world's oldest & largest OS-9 software house!

CUTS PROGRAMMING TIME UP TO 80%

6809/68000-68030 Save 90%

SCULPTOR-a 4GL - Only from S.E. Media at these prices. OS-9 levels one and two (three GIMIX) 6809, all 68XXX OS-9 standard systems. Regular SCULPTOR versions 1.4:6. One of if not the most efficient and easy to develope DBMS type systems running under OS-9! A system of flexible keyed file access that allows extremely fast record and data retrieval, insertion and deletion or other programmed modifications. Access by key or in ascending order, very fast. The system provides automatic menu generation, compilation and report generation. Practically unlimited custom input format and report formatting. A rich set of maintenance and repair utilities. An extremely efficient development environment that cuts most programming approximately 80% in development and debugging! Portable, at source level, to MS-DOS, UNIX and many other languages and systems.

Standard Version: 1.6 6809 - $1295.00
68000 $1295.00
68020 $1990.00

Due to a "Special One Time" Purchase, We Are Making This Savings Offer. Quantities Limited!

Once this supply is gone the price goes back up!

System OS-9: 6809/68000-68030

• Regular ~~$1295.00~~

ONLY **$99.95**

S.E. MEDIA
POB 849 5900 CASSANDRA SMITH ROAD
HIXSON, TN 37343 615 842-4601
TELEX 510 600-6630

$AVE - WHILE SUPPLIES LAST!

C User Notes

The C Programmers Reference Source. Always Right On Target!

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter presents an enhanced version of the getopt function, which is used to scan the command line. This function was originally placed in the public domain by someone unknown, but was later enhanced by Keith Bostic and others, and subsequently extended and provided by

Lloyd Zusman
Master Byte Software
Los Gatos, California

who also provided much of the text below.

COMMAND-LINE OPTION SCAN FUNCTION

Lloyd calls the enhanced version of getopt() by the name egetopt(). The default behavior of this routine is the same as that of getopt(), but it has some optional features that make it more useful.

These options are controlled by the settings of some global variables. By not setting any of these extra global variables, they have the same functionality as getopt().

The egetopt() function acts like getopt() with the following enhancements:
The '?' which gets returned when there is an unrecognized option is now stored in a global integer called 'optbad', and the caller can set this value to anything. The initial value in 'optbad' is '?', which means that the default behavior is just like that of getopt().

For example, if egetopt() is to return '~' instead of '?' when it encounters an invalid option, the following lines should be executed before egetopt() is called:

```
extern int optbad;
optbad = (int)'~';
```

Options can begin with characters other than just '-'. There is now a global character pointer called 'optstart'. It points to a string which consists of a list of characters which can be used to begin options. The initial string that 'optstart' points to is "-", so the default behavior is like that of getopt().

For example, to allow both '+' and '-' as option delimiters, put the following lines in the code before egetopt() gets called:

```
extern char *optstart;
optstart = "-+";
```

Now that there's a choice of the characters that can precede options it's desirable to let the caller know what character begins a given option. In egetopt(), the global integer 'optchar' will now contain the character that begins a given option, or 0 if there was an error. Insert the following declaration line and check the value of 'optchar' after each call to egetopt():
extern int optchar;
The original getopt() writes error messages to file descriptor 2 (or to stderr, depending on the implementation). In egetopt(), this file descriptor may point to any desired file. The global integer 'opterrfd' contains the file descriptor to use for writing error messages; it is initialized to 2.

As an example, to send egetopt() errors to go to the file "egetopt.errs", use code similar to the following before calling egetopt():

```
extern int opterrfd;
FILE *eout = fopen("egetopt.errs", "w");

if (!eout)
{

/* error condition/

exit(1);

}

opterrfd = fileno(eout);
```

Some implementations of getopt() allow the setting of the global integer 'opterr' to control whether error output is printed: it is initialized to 1, which enables error output (as does any non-zero value); setting it to 0 disables error output. In egetopt(), 'opterr' is treated the same way.

The old getopt() forces you to use ':' in the string of option letters to show that a given option takes an argument. There is now a global integer called 'optneed' which contains this value, it may be changed; 'optneed' is initialized to ':'.

In addition, getopt() is unable to handle optional option arguments. For example, if an option called 'd' was specified as taking an argument to the program 'foo', getopt() would return the following results when invoking 'foo' in different ways:

```
1)      foo -dABC -x ...
getopt() return:        'd'
optarg:                 "ABC"
2)      foo -dABC -x ...
getopt() return:        'd'
optarg:                 "ABC"
3)      foo -d -x ...
A)      getopt() return:        'd'

optarg:                 "-x"
B)      getopt() return:        'd'

optarg:                 NULL
```

In the case of number 3, sometimes one would prefer to get the latter results. This would allow "-x" to be handled as another option in the next call. In the old getopt(), the 3B behavior may be gotten by testing the first character of 'optarg' and decrementing 'optind' if this character is '-'. The new routine may have either behavior directly.

Since this behavior isn't always desired, egetopt() checks a global integer called 'optmaybe' which allows the programmer to control whether an option with an argument will get treated as number 3A or as number 3B above. It is used similarly to 'optneed'. It is initialized to 0, meaning that behavior 3B is impossible by default.

The following example shows how 'optneed' and 'optmaybe' can be used:

```
extern int optneed;
extern int optmaybe;

optneed = (int)'!';  /* use '!' instead of ':' */
optmaybe = (int)'%'; /* use '%' for optional args
*/
```

```
while ((c = egetopt(argc, argv, "abc!d%x")) != EOF)
...
```

In this example, options 'a', 'b', and 'x' take no arguments, option 'c' takes a mandatory argument, and option 'd' takes a non-mandatory argument. If this is contained in program 'foo', the following behavior will be observed when it it run:

```
foo -a -cABC -dXYZ -d -x -c -b ...
egetopt() return:       'a'
optarg:                 NULL

egetopt() return:       'c'
optarg:                 "ABC"

egetopt() return:       'd'
optarg:                 "XYZ"
>>>>>>>>>>      egetopt() return:       'd'
>>>>>>>>>>      optarg:                 NULL
>> NOTE >>
>>>>>>>>>>      egetopt() return:       'x'
>>>>>>>>>>      optarg:                 NULL
egetopt() return:       'c'
optarg:                 "-b"
```

Remember that 'optneed' is initialized to ':' and 'optmaybe' is initialized to 0. This causes behavior identical to that of getopt() unless specifically overridden.
Since the default behavior of egetopt() is the same as that of getopt(), there is no reason why it cannot be renamed to getopt() and used it in place of the original. It was given a new name so as not to conflict with the name of the original getopt() function.

Following is the source code for egetopt().

```
/*
 * egetopt.c - Extended 'getopt'.
 *
 * Original version:
 *
 *    Keith Bostic
 *
 * Current version:
 *

 *    Lloyd Zusman
 *    Master Byte Software
 *    Los Gatos, California
 *
 */

#ifndef EOF
# define EOF        (-1)
#endif /* ! EOF */

#ifndef NULL
# define NULL       (char *)0
#endif /* ! NULL */

/*
```

```
 * None of these constants are referenced in the executable portion of
 * the code ... their sole purpose is to initialize global variables.
 */
#define BADCH       (int)'?'
#define EMSG        ""
#define ERRFD       2
#define MAYBESEP    (int)'\0'
#define NEEDSEP     (int)':'
#define START       "-"

/*
 * Here are all the pertinent global variables.
 */
char *optarg;           /* argument associated with option */
char *optstart = START; /* list of characters that start options */
int optbad = BADCH;     /* character returned on error */
int optchar = 0;        /* character that begins returned option */
int opterr = 1;         /* if true, output error message */
int opterrfd = ERRFD;   /* file descriptor for error text */
int optind = 1;         /* index into parent argv vector */
int optmaybe = MAYBESEP; /* flag for optional argument */
int optneed = NEEDSEP;  /* flag for mandatory argument */
int optopt;             /* character checked for validity */
/*
 * Conditionally print out an error message and return.
 */
#define TELL(S)     { \
    if (opterr && opterrfd >= 0) \
    { \
        char option = optopt; \
        write(opterrfd, *nargv, strlen(*nargv)); \
        write(opterrfd, (S), strlen(S)); \
        write(opterrfd, &option, 1); \
        write(opterrfd, "\n", 1); \
    } \
    return (optbad); \
}

/*
 * _sindex works similarly to index() and strchr().
 */
static char *
_sindex(string, ch)
char *string;
int ch;
{
    if (string)
    {
        for ( ; *string; ++string)
        {
            if (*string == (char)ch)
            {
                return (string);
            }
        }
    }

}
/*
 * Extended getopt function
 */
int
egetopt(nargc, nargv, ostr)
```

```
int nargc;
char **nargv;
char *ostr;
{
    static char *place = EMSG;     /* option letter processing */
    register char *oli;            /* option letter list index */
    register char *osi = NULL;     /* option start list index */

    if (!nargv)
        return (EOF);

    if (nargc <= optind || !nargv[optind])
        return (EOF);

    if (!place)
        place = EMSG;

    /* Update scanning pointer.  */
    if (!*place)
    {
        place = nargv[optind];
        if (!place)
            return (EOF);
        if (osi = _sindex(optstart, *place))
            optchar = (int)*osi;
        if (optind >= nargc || !osi || !*++place)
            return (EOF);

        /*
         * Two adjacent, identical flag characters were found.
         * This takes care of "--", for example.
         */
        if (*place == place[-1])
        {
            ++optind;
            return (EOF);
        }
    }

    /*
     * If the option is a separator or the option isn't in the list,
     * we've got an error.
     */
    optopt = (int)*place++;
    oli = _sindex(ostr, optopt);
    if (optopt == optneed || optopt == optmaybe || !oli)
    {
        /*
         * If we're at the end of the current argument, bump the
         * argument index.
         */
        if (!*place)
            ++optind;
        TELL(": illegal option -- ");     /* byebye */
    }

    /*
     * If there is no argument indicator, then we don't even try to
     * return an argument.
     */
    if (!*++oli || (*oli != optneed && *oli != optmaybe))
    {
        /*
```

```
         * If we're at the end of the current argument, bump the
         * argument index.
         */
        if (!*place)
            ++optind;
        optarg = NULL;
    }
    /*
     * If we're here, there's an argument indicator.  It's handled
     * differently depending on whether it's a mandatory or an
     * optional argument.
     */
    else
    {
        /*
         * If there's no white space, use the rest of the
         * string as the argument.  In this case, it doesn't
         * matter if the argument is mandatory or optional.
         */
        if (*place)
            optarg = place;
        /*
         * If we're here, there's whitespace after the option.
         *
         * Is it a mandatory argument?  If so, return the
         * next command-line argument if there is one.
         */
        else
        if (*oli == optneed)
        {
            /*
             * If we're at the end of the argument list, there
             * isn't an argument and hence we have an error.
             * Otherwise, make 'optarg' point to the argument.
             */
            if (nargc <= ++optind)
            {
                place = EMSG;
                TELL(": option requires an argument -- ");
            }
            else
                optarg = nargv[optind];
        }
        /*
         * If we're here it must have been an optional argument.
         */
        else
        {
            if (nargc <= ++optind)
            {
                place = EMSG;
                optarg = NULL;
            }
            else
            {
                optarg = nargv[optind];
                if (!optarg)
                    place = EMSG;
                /*
                 * If the next item begins with a flag
                 * character, we treat it like a new
                 * argument.  This is accomplished by
                 * decrementing 'optind' and returning
```

```
                 * a null argument.
                 */
                else
                if (_sindex(optstart, *optarg))
                {
                    --optind;
                    optarg = NULL;
                }
            }
        }
        place = EMSG;
        ++optind;
    }
    /*
     * Return option letter.
     */
    return (optopt);
}
```

EXAMPLE C PROGRAM

Following is this month's example C program; it is a test driver for the command-line processing function egetopt() just described.

```
#include <stdio.h>

/*
 * This is a program for demonstrating
 * the capabilities of egetopt().
 * Run it with various combinations of
 * options and arguments on the command
 * line to see how egetopt() works.
 *
 */

#define OPT_STRING    "abc~d~e?f?"
/* Meaning:
 *
 * -a and -b take no arguments.
 * -c and -d take mandatory arguments
 * -e and -f take optional arguments
 */

#define OPT_CHARS     "-+="
/* Meaning:
 *
 * Options can begin with '-', '+', or '='.
 */

/*
 * New global variables used in egetopt() only:
 */
/* string which contains valid option start chars */
extern char *optstart;
/* what egetopt() returns for a bad option */
extern int optbad;
/* character which begins a given argument */
extern int optchar;
/* where egetopt() error messages go */
extern int opterrfd;
/* character used for optional arguments */
extern int optmaybe;
/* character used for mandatory arguments */
extern int optneed;
```

```
/*
 * Global variables which exist in getopt() and egetopt():
 */
/* the argument of the option */
extern char *optarg;
/* set to 0 to suppress egetopt's error messages */
extern int opterr;
/* index of current argv[] */
extern int optind;
/* the actual option pointed to */
extern int optopt;

main(argc, argv)
int argc;
char **argv;
{
    int ch;

    /* errors to stdout */
    opterrfd = fileno(stdout);
    /* set this to 1 to get egetopt's error msgs */
    opterr = 0;
    /* return '!' instead of '?' on error */
    optbad = '!';
    /* mandatory arg identifier (in OPT_STRING) */
    optneed = '~';
    /* optional arg identifier (in OPT_STRING) */
    optmaybe = '?';
    /* characters that can start options */
    optstart = OPT_CHARS;

    while ((ch = egetopt(argc, argv, OPT_STRING)) != EOF)
    {
        printf("\n\toption index (optind) after egetopt(): %5d\n", optind);
        printf("\t\tegetopt() return value:            %c (%d)\n", ch, ch);
        printf("\t\tchar that begins option (optchar): %c\n", optchar);
        printf("\t\tactual char looked at (optopt):    %c\n", optopt);
        printf("\t\toption argument:                   \"%s\"\n",
            !optarg ? "(null)" : optarg);
    }

    for ( ; optind < argc; ++optind)
    {
        printf("\n\targument index                        %5d\n", optind);
        printf("\t\targument:                          \"%s\"\n",
            !argv[optind] ? "(null)" : argv[optind]);
    }

    exit(0);
}

EOF
```

SOFTWARE USER NOTES

A Tutorial Series

By : Ronald W Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

From Basic Assembler to HLL's

RBASIC

Since writing the last column, I have spent some time looking at RBASIC. I've found a few minor bugs and sent the report to Bob Jones. Bob has written me a letter and sent a couple of revisions. I've been working on coding some scientific functions that would be accurate to 19 digits, and I've sent the first few off to Bob. Much of the information I sent him has crossed his first letter in the mail. Unfortunately the mail from Michigan to Brittish Columbia Canada takes about a week. Bob reported that he is interested in increasing the accuracy of the scientific functions to match the math package, and we will most likely colaborate on some improvements.

I continue to be pleased with RBASIC. I did some comparison tests with TSC Extended BASIC running on a 6809 (2 MHz). The PT68K-2 68000 with RBASIC runs on the average three to four times faster than the 6809 system doing rather math intensive calculations.

I gathered from Bob's letter that he badn't expected much of a response from me, and that he was glad for some feedback. I mentioned having started on his logic series and he said that I was only the second person who had responded in any way. Come on, readers. If you like something in '68' Micro Journal, let the author hear from you. It is discouraging to work at something month after month and only see one or two responses. Write Micro Journal or the author of the material that you would like to see continue, and it will have a better chance of doing so.

Having looked over the material from Bob more thoroughly, I will say that there are a few things in RBASIC that are not completely compatible with XBASIC, but the differences are mostly due to the processor differences. For example the USR function can't be the same. I won't elaborate in great detail because most of the differences are in things generally not used extensively in BASIC programs. I've urged Bob to add a few new features to RBASIC that will be extensions and will not destroy compatibility with old XBASIC programs. For example, when XBASIC first was done and I looked at it, I was disappointed to find a couple of the nice features of Bob Uiterwyk's old BASIC missing. To write three data items to a file, separated by commas, for example, in the old BASIC one simply had to:

```
100 WRITE #1, A,B,C
```

TSC's BASIC was made compatible with a BASIC from DEC, and it lacks the WRITE statement completely. Instead, as you BASIC users know, you had to:

```
100 PRINT #1, A;",";B;",";C
```

That is, you have to explicitly code the printing of the commas between the data items. I can certainly see having the PRINT statement work the same way to a file as it does to a printer, but there needs to be an easier way to write data to a file. Bob indicated that he may well be interested in a number of enhancements in the future. The other thing I have always thought was missing from XBASIC is some sort of end of file test for input files.

```
90 INPUT #1,A$(N%)
100 IF EOF(1) THEN 120
105 N%=N%+1
110 GOTO 90
120 CLOSE 1
```

Rather, with XBASIC you have to trap errors and test to see if the error is #8.

```
10 ON ERROR GOTO 1000
.
.
90 INPUT #1,A$(N%)
100 N%=N%+1
110 GOTO 90
120 CLOSE 1
.
.
1000 IF ERR=8 AND ERL=90 THEN RESUME
120 ELSE ONERROR GOTO 0
```

If you had a program with a number of files that could reach end of file at various places within the program, you would have to have a number of lines or "cases" for ERR, the error number and ERL, the line on which the error occurred, with different RESUME lines, and a catch all ONERROR GOTO 0 at the end so that other errors than end of file will be reported properly when debugging the program. It has always seemed to me as though BASIC should provide such a function so the user doesn't have to build the error trap into even a simple program to read a data file and print the contents to the screen. On the other hand, I suppose we each have our little pet "It would only take a few lines of code to add..." feature, and if Bob were to add all of them RBASIC would be big and cumbersome. Bob, I still hope you will add "mine".

A few weeks have passed since I wrote the above. Bob Jones has fixed the few little bugs that I found in RBASIC and we are continuing our discussions of how to make the math more efficient. In fact, the floating point multiply routine has been improved to be about 4.25 times faster. I'll have to scrap the few benchmarks I've run so far and wait for things to settle down a little more. Let me say that this is a very complete BASIC having full PRINT USING features, sequential and random file access, etc. I'll do a full review on it in the future. If you use BASIC for quick and dirty programs, exploratory programming, or solving problems that are very complex, you will want this BASIC.

Assembler

I received a call from a reader the other night, who had a few problems that might be common to some others of you who are just starting out on the 68000 and are new to Assembly language. I will include a short listing of a program to set an Epson printer to the 132 column (16.7CPI) mode. There are several things to point out about it. The caller's first problem was that he had written and assembled a short program but when he tried to run it SK*DOS gave him an error message that said that there was no transfer address. The transfer address is the address in the program at which it is to begin to execute. The transfer address need not necessarily be the first address in the program so the system needs some way of telling from the program code on the disk where to start executing the program relative to its first load address. To give it this information, the "start executing" address of the program must have a label. In the case of listing 1, it is the label START. Now at the end of the program the END statement tells the assembler that there is no more program to code. After the word END, you simply place the starting label, START in this case. When the program is assembled and saved to the disk as a binary file, the transfer address is coded into the file from this information. Otherwise, the code is treated as a fragment, perhaps part of another program that is to be accessed from elsewhere, and it has no transfer address, and so cannot be run by SK*DOS.

It is now about two weeks since I wrote the above. I've been plagued by a lot of work at work and some chores that had to get done around the house. I recently found a bug in my DDUMP utility published here a few months ago. DDUMP, you may remember, does a HEXADECIMAL and ASCII dump of a disk file, sector by sector. I had used it primarily to dump binary files, but one day not too long ago, I decided to use it to look for something funny in a text file. I found that it exited after the first sector. I made a few changes and then found that it insisted on repeating the first sector 5 times and then went on to the next one. After a little hair pulling I discovered that it would foul up if there happened to be a TAB at the start of the sector.

Text files in SK*DOS use a technique called space compression. Rather than put 10 or 12 $20 (space) characters in a file, SK*DOS automatically compresses consecutive spaces by using a horizontal tab character $09 followed by a space count. If there are two spaces together, of course, you come out even, using two characters. However, any number from 3 to 255 spaces can be compressed to two characters by this method. What I eventually figured out, is that the FSKIP function of SK*DOS doesn't perform its function if FREAD has gotten the system into the middle of a space expansion while reading the file. FSKIP is supposed to skip the remainder of the current sector so the next sector will be read on the next FREAD call. It worked fine except when in the middle of a space expansion. I decided to tell SK*DOS not to expand spaces, since I wasn't using the read function to get the data, but was taking it directly from the file control block. All that is needed is to stuff $FF into the 59th byte of the file control block after opening the file but before reading any data. While I was looking at it I cleaned up the code a little. The listing of version 2.0 is included here. I've improved the error handling and trapped error #8 as end of file. That error occurs when you try to read

past the last valid sector of a file. Essentially I have done what I was complaining about having to do in BASIC above.

I have a few copies of the SK*DOS version of PAT, my editor out for testing. I have held off on distributing it because I wanted to check it out personally on a PT68K-2 system using a monitor and IBM keyboard. From one source I hear how slow that system is compared to a terminal, and from another I hear how fast it is. I want to check it out for myself and perhaps write a driver that will allow some fancies on the screen. My other motive for this is that Dan Farnsworth of Palm Beach Software has sent me his EDDI text editor that runs with this setup. I have ordered the monochrome board and IBM clone keyboard for my system. I have a monitor kicking around somewhere, so I will be able to run the system in that mode and free up a terminal. Actually I have still another motive, that of the possibility of doing some graphics for my work projects. That is a long term goal.

I've been using RBASIC for several days to analyze some peculiar problems at work, and have not found any further bugs in it. It is nice to have a BASIC that runs on the 68K systems for testing ideas and techniques for my work projects. Along that line, I have a most peculiar problem with combinations of weights. Essentially I can have 5 different weights but I can use only three or less weights at any one time. I can use one of the weight values repeatedly as in 7,2,2 etc. The problem is to cover the greatest range of weight in unit steps with various combinations of these weights. I found a set consisting of 1, 2, 3, 7, and 18 that would let me make combinations of three or less weights for values from 1 to 28 with no values skipped. Many of the combinations are redundant, and I would guess that there would be other sets that might span a larger range. I am perplexed that I could find no logical approach to calculating the best set of weights. Of course if there were not a limitation on the number of weights a simple binary progression would do the job.

In a similar problem I have 6 weights to be taken 1 2 or 3 at a time. I came up with 1, 2, 3, 7, 11, and 26, which gets me 0 to 40 in steps of 1. Are there any mathematicians or logicians out there who can give me a clue as to how to approach this problem to get the largest coverage from a given set of weights? A set of 7 taken 3 at a time get me from 0 to 52. They are 1, 2, 3, 7, 11, 15, and 34. If any of you can figure out a good series that will go farther in continuous steps of 1 I'd like to hear about it. The matter is now academic because of other limitations on the problem, but it is still of interest.

Last Minute News

I just received a new version of RBASIC in which the floating point divide has been speeded up by more than a factor of 2, and the display of integer numbers has been cleaned up. The original had a problem if the user specified too many digits for display. It would sometimes display with a slight conversion error showing 15.99999999999999978 rather than the actual value of 16 for example. Bob has cured that problem in this latest version as well.

I just returned from a vacation trip with my wife to New Zealand with a stop in Hawaii on the return trip. We visited John Spray in Auckland and did a whirlwind tour of the country. I can say that it is truly a beautiful place. We thoroughly enjoyed our visit there. I had the opportunity to drive a right-hand drive vehicle on the left side of the road. That wasn't as difficult as it might seem, but I did have a hard time with the manual transmission shift lever on the left and turn signal on the right. I kept trying to shift gears with the turn signal. We found a strange mixture of prices. Gasoline (or Petrol) costs about twice what it does here in the U.S. but a good dinner in a restaurant about half as much.

More to the point of this column, John and I visited several computer stores while we were there. Most of them are just like Computerland and their clones with slick equipment setups and sales people to tell you all about the advantages of their products. We did visit one that was explicitly for the "hacker". They had computer boards laid out on tables and very low prices. We found out that many of the boards were not in working condition and a purchaser could either buy a tested one at a much higher price or buy two or three in hopes of getting one that works or can be fixed. There were also quantities of used disk drives and the like for sale under the same terms. I'd say that computing there has gone pretty much down the IBM and clone road just as it has here. Outside of the industrial control area there is very little else around.

Well, the trip has taken three weeks of my time so this material is about to be late. Larry Williams called this morning to see if I had any material ready to mail in, so I guess I'll run this through the appropriate conversion program and add the two listing files to a disk so it can be mailed on time.

```
                    **************************************************
                    * DISKFILE DUMP PROGRAM                          *
                    *                                                *
                    * COMMANDS:                                      *
                    * N XXXX NEXT PAGE TO BE DUMPED                  *
                    * B BACK A SECTOR                                *
                    * F FORWARD A SECTOR                             *
                    **************************************************

                    * SK*DOS / 68K EQUATES FOR USER PROGRAMS

          0000A029  GETCH   EQU     $A029           Get input character with echo (Alts)
          0000A023  GETNAM  EQU     $A023           Get filename into FCB
          0000A024  DEFEXT  EQU     $A024           Set default extension
          0000A005  FOPENR  EQU     $A005           Open file for read
          0000A001  FREAD   EQU     $A001           Read a byte
          0000A011  FSKIP   EQU     $A011           Next Sector
          0000A031  TOUPPR  EQU     $A031           Conv't char in D5 to Upper Case
          0000A02F  HEXIN   EQU     $A02F           Input hexadecimal number
          0000A03A  OUT2H   EQU     $A03A           Output 2 hex digits
          0000A03B  OUT4H   EQU     $A03B
          0000A03C  OUT8H   EQU     $A03C           Output 8 hex digits
          0000A034  PCRLF   EQU     $A034           Print CR/LF
          0000A037  PERROR  EQU     $A037
          0000A036  PNSTRN  EQU     $A036           Print string (Without CR/LF)
          0000A035  PSTRNG  EQU     $A035           Print CR/LF and string
          0000A033  PUTCH   EQU     $A033           Output character
          0000A000  VPOINT  EQU     $A000           Point to SK*DOS variable area
          0000A01E  WARM$T  EQU     $A01E           Warm start

                    *
                    *
000000 A000         START   DC      VPOINT          GET POINTER
000002 284E                 MOVE.L  A6,A4           FCB POINTER
000004 A023                 DC      GETNAM          GET FILENAME
000006 6500 0090(00098      BCS     HELP
00000A 7801                 MOVE.L  #1,D4           CODE FOR .TXT
00000C A024                 DC      DEFEXT
00000E A005                 DC      FOPENR
000010 197C 00FF 003B       MOVE.B  #$FF,59(A4)     SET NO SPACE COMPRESSION
000016 A001         LOOP    DC      FREAD
000018 6618     (00032      BNE.S   ERROR
00001A 204C                 MOVE.L  A4,A0
00001C D1FC 0000 0060       ADD.L   #96,A0          POINT AT SECTOR INFO
000022 611C     (00040      BSR.S   OPAGE
000024 A029                 DC      GETCH
000026 A031                 DC      TOUPPR
000028 0C05 0045            CMP.B   #'E',D5
00002C 6710     (0003E      BEQ.S   EXIT
00002E A011         CONTIN  DC      FSKIP
000030 60E4     (00016      BRA     LOOP
000032 0C2C 0008 0001 ERROR CMP.B   #8,1(A4)        IS END OF FILE?
>000038 6700 0004(0003E     BEQ     EXIT
00003C A037                 DC      PERROR
00003E A01E         EXIT    DC      WARMST
                    *
                    * ROUTINE TO OUTPUT A PAGE IN HEX AND ASCII
                    *
000040 A034         OPAGE   DC      PCRLF
000042 4240                 CLR.W   D0              LINE COUNTER
000044 A034                 DC      PCRLF
                    * LOOP FOR LINES
000046 323C 000F    LLOOP   MOVE.W  #15,D1          COUNTER FOR CHARACTERS
00004A 1800                 MOVE.B  D0,D4
00004C E904                 ASL.B   #4,D4           ADDRESS OF FIRST BYTE OF LINE ON PAGE
00004E A03A                 DC      OUT2H
000050 183C 0020            MOVE.B  #$20,D4
000054 A033                 DC      PUTCH           SPACE
000056 A033                 DC      PUTCH           SECOND SPACE
                    * INSIDE LOOP FOR 16 CHARACTERS IN HEX
000058 1818         CLOOP   MOVE.B  (A0)+,D4
00005A A03A                 DC      OUT2H           OUTPUT FIRST BYTE
00005C 183C 0020            MOVE.B  #$20,D4
```

```
000060 A033                     DC      PUTCH           SPACE
000062 57C9 FFF4(00058          DBEQ    D1.CL  P        CHARACTERS
                        * NOW DO ASCII CHARACTERS, "." FOR NON PRINTABLE
000066 183C 0020                MOVE.B  #$20,D4
00006A A033                     DC      PUTCH           EXTRA SPACE BEFORE ASCII
00006C 91FC 0000 0010           SUB.L   #16,A0
000072 323C 000F                MOVE.W  #15,D1          RELOAD COUNTER FOR CHARACTERS
000076 1818             ALOOP   MOVE.B  (A0)+,D4
000078 0244 007F                AND     #$7F,D4         MASK OFF HI ORDER BIT
00007C 0C04 0020                CMP.B   #$20,D4         IS IT PRINTABLE?
000080 6C04     (00086          BGE.S   AL1             IF YES
000082 183C 002E                MOVE.B  #'.',D4         ELSE PRINT PERIOD
000086 A033             AL1     DC      PUTCH
000088 57C9 FFEC(00076          DBEQ    D1,ALOOP
00008C A034                     DC      PCRLF
00008E 5240                     ADD     #1,D0
000090 0C00 0010                CMP.B   #16,D0
000094 66B0     (00046          BNE     LL  P           NEXT LINE
000096 4E75                     RTS

000098 49FA 0006(000A0 HELP     LEA     HLPMSG(PC),A4
00009C A035                     DC      PSTRNG
00009E A01E                     DC      WARMST

0000A0 5379 6E74 6178  HLPMSG   DC.B    "Syntax: DDUMP FILENAME   Defaults are drive",$0D,$0A
0000D2 616E 6420 2E54           DC.B     "and .TXT extension.",$0D,$0A,$0A
0000E8 4444 554D 5020           DC.B    "DDUMP dumps a disk file to the terminal or",$0D,$0A
00011C 6174 2061 2074           DC.B    "at a time.  The dump displays 16 bytes in HEX followed by",$0D,$0A
000157 7468 6520 4153           DC.B    "the ASCII representation of the same 16 bytes.Non-",$0D,$0A
00018D 7072 696E 7461           DC.B    "printable characters are displayed as  periods.",$0D,$0A
0001BD 4174 2074 6865           DC.B    "At the command prompt, F will cause the(forward)",$0D,$0A
0001F5 7365 6374 6F72           DC.B    "sector to be displayed, and B (back) will display",$0D,$0A
00022C 7072 6576 696F           DC.B    "previous sector.  E will Exit the program to",$0D,$0A
00025F 4620 7768 656E           DC.B    "F when the last sector of the file has reached.",$0D,$0A,$04

                                END     START
  0 ERRORS DETECTED
```

```
                        * PROGRAM TO SET EPSON TO 16 CPI
                        *
          0000A033      PUTCH   EQU     $A033
          0000A000      VPOINT  EQU     $A000
          0000A01E      WARMST  EQU     $A01E
          00000CCB      DEVOUT  EQU     3275
000000                          ORG     $0000
000000 A000             START   DC      VPOINT
000002 1D7C 0002 0CCB           MOVE.B  #2,DEVOUT(A6)
000008 183C 000F                MOVE.B  #$0F,D4
00000C A033                     DC      PUTCH
00000E 1D7C 0000 0CCB           MOVE.B  #0,DEVOUT(A6)
000014 A01E                     DC      WARMST
                                END     START

  0 ERRORS DETECTED
```

+++

Logically Speaking

The Mathematical Design of Digital Control Circuits

Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

SOLUTIONS TO TEST FOURTEEN-A

Relays Operated	Code	X=0	X=1	Z
0	0	0	1	
1L	1	2	3	1
S1	2	2	-	1
2L	3	-	4	
3L	4	5	-	1
S3	5	5	-	1

	X=0	X=1	Z
0	0	1	
1	2	3	1
2	2	-	1
3	-	2	

Combine 2,4,5

6 contacts
9 springs

(a)

Relays Operated	Code	X=0	X=1	Z
0	0	0	1	
1L	1	2	3	1
S1	2	2	1	1
2L	3	-	4	
3L	4	5	-	1
S3	5	5	1	1

	X=0	X=1	Z
0	0	1	
1	2	3	1
2	2	1	1
3	-	4	
4	2	-	1

Combine 2 ,5

8 contacts
13 springs

(b)

Relays Operated	Code	X=0	X=1	Z
0	0	0	1	1
1L	1	2	3	1
S1	2	2	1	1
2L	3	-	4	
3L	4	5	-	1
S3	5	5	1	1

	X=0	X=1	Z
0	0	1	1
1	0	2	1
2	-	3	
3	0	-	1

Combine 0,2,5 renumber 3,4 → 2,3

6 contacts
10 springs

(c)

Relays Unoperated	Code	X=0	X=1	Z
0	0	1	0	
1L	1	2	-	
2L	2	-	3	1
S2	3	-	3	1

	X=0	X=1	Z
0	1	0	
1	2	-	
2	-	2	1

Combine 2,3

4 contacts
6 springs
(h)

Be careful here! Note that we're keeping track of the number of UNOPERATED relays.

Relays Operated	Code	X=0	X=1	Z
0	0	0	1	
1L	1	2	3	1
S1	2	2	-	1
2L	3	4	-	1
S2	4	4	-	1

	X=0	X=1	Z
0	0	1	
1	2	2	1
2	2	-	1

Combine 2,3,4

3 contacts
5 springs
(i)

This is the inverse of the specs, and should be graphically complemented to give the desired output conditions. NOTE : Line-1 needs no contacts.

Relays Operated	Code	X=0	X=1	Z
0	0	0	1	
1L	1	2	3	
S1	2	2	4	
2L	3	5	6	
S1 +1L	4	-	7	
S2	5	5	8	
3L	6	2	6	
S1 + 2L	7	9	-	1
S2 + 1L	8	9	10	1
S1 + S2	9	9	-	1
S2 + 2L	10	-	11	
S2 + 3L	11	9	11	1

	X=0	X=1	Z
0	0	1	
1	2	3	
2	2	4	
3	5	6	
4	-	7	
5	5	8	
6	2	6	
7	7	-	
8	7	9	
9	-	10	
10	7	10	

Combine 7,9
Renumber 10,11 → 9,10

19 contacts
29 springs
(j)

NOTE : In line-6, column X=0, I've not opened up a line for S3, because this circuit is interested in sets of TWO only. Any other size is all the same to this network, so I can save myself a lot of work by naking all sets larger or smaller than TWO take on the same name, namely S1, which I now choose to interpret as "One set of any size except TWO". Similarly in Line-11.

If you attempted each and every one of the above, especially the last five (which are reasonably tough), you should have a really good idea how to create iterative tables by now! They were good practice, anyway! So, if everyone's ready, let's look at a final high-powered technique for manipulating a prototype table.

Mile 18 - heading for Mile 19

MULTIPLE ASSIGNMENT

In our earlier study of iterative networks we've always produced a prototype cell with one input-line for power, which is switched from one level to another as it proceeds through the cascaded network. Eventually, if it's not cut off along the way, power arrives at one or more output-lines which are connected together.
Let's pretend we've developed a table which switches between eight different levels, numbered from 0 through 7, where each level represents a different piece of info being passed from one relay to the next along our chain of "n" relays. I don't think I'm far wrong when I say that all of you know that eight different numbers (or data-codes) CAN be transmitted in binary along only THREE wires, instead of the EIGHT original lines. This is just what we're going to attempt in our final session with iterative networks. We're going to "code" our info, using multiple line assignments, instead of our usual "one line = one piece of info". Of course we're going to this scientifically, and develop codes which won't create any conflict one with the other. And just so we can tell whether "multiple" coding of a table has produced any improvement over "single" coding, we obviously have to have some standard of comparison.

SPRINGS AND CONTACTS

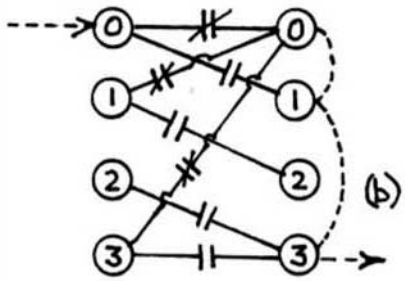
This will be based on the number of "springs" and "contacts" in both implementations. You already know what a "contact" is, so I'll simply define a "spring" as a terminal on a relay-contact to which a wire can be connected. An ordinary NO-contact or NC-contact has TWO springs, because two wires can be connected to it, one at each end of the contact. A transfer-contact, on the other hand, has THREE springs, because one wire can be fastened to the common centre-point, and one each to the NO and NC contacts between which it can be switched.
So without further ado, let's get on with a new set of specifications with a slightly different twist to vary our diet a little, and compare the two methods of assignment. Here then is

ITERATIVE NETWORKS - MULTIPLE ASSIGNMENT - EXAMPLE 1

We're called on to design the prototype cell for a network of "n" relays whose output-line is normally powered-up, but which has to shut off iff exactly ONE set of TWO relays becomes operated.
This time I don't think you'll need a step-by-step explanation of the table shown in Diagram 96a, especially if you take time out to actually create it yourself. Just a brief pointer to the Z-column, where you'll notice that we're calling for power in all lines except where the set of 2 relays is formed. The normal table is necessary as the first step in our new procedure, and I've already drawn the prototype cell in 96b for comparison purposes later.

Relays Operated	Code	X 0	X 1	Z
0	0	0	1	1
1L	1	0	2	1
2L	2	-	3	
3L or more	3	0	3	1

(a)



(b)

Diagram 96

If we look at the network, we see that X has a total of seven contacts, which, when grouped to form transfer-contacts, require eleven springs to form the circuit - - 3 transfers of 3 springs each, and 1 NO-contact of 2 springs. With me so far? Good!
Note that our prototype table shows than an output is required for lines, or states, 0, 1 and 3 (which is why outputs 0, 1 and 3 are joined together in 96b), and it's at this point that we begin our multiple assignment coding.

CREATING A MULTIPLE-ASSIGNMENT PROTOTYPE TABLE

This begins with Diagram 97a, where our original "Code" column has now been moved out to the extreme left. Keep in mind that Z calls for an output in lines 0, 1 and 3 - highest number therefore being "3" - so everywhere that 0, 1 or 3 appears in our original table (in the "code" column or either of the X-columns) we're going to write "3" (the highest number called by Z).

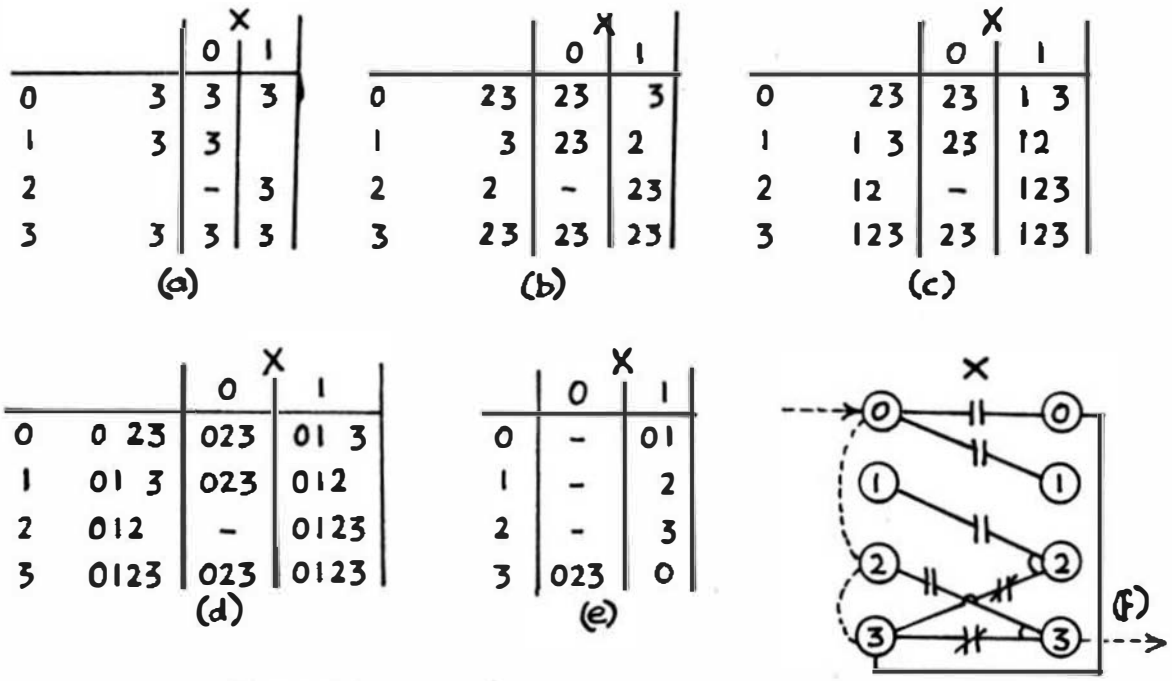


Diagram 97

The procedure we're going to follow isn't difficult, but we MUST be careful not to make any mistake as we go along. It's a continuous repeat of a very simple step, so stay close and it'll all come out OK. Still with 97a then, we've just set out a new coding, both in the "Code" column and in both internal columns, and at the moment it seems that we have 3s everywhere. Now, treading very warily, let's look at column X=0 and ask ourselves "For every 3 in this column, is there a consistent digit in the code-column?" We see that this IS so, namely that for every 3 in this column there's a 3 externally in the code-column. OK then, let's move to the X=1 column and ask the same question. This time we have to answer "No", because, although the 3s in lines 0 and 3 both line up with an external 3, the 3 in line-2 doesn't even HAVE an external number, let alone one which is the same for ALL the 3s in this column.

ADDING CODE-2

We'll rectify this situation at once, by inserting IN THE CODE-COLUMN the next lowest number (ie, 2) directly opposite the 3s in column X=1 (see Diagram 97b). So now we DO have a consistent number which lines up with ALL our 3s - - namely, an external 3 lining up with the 3s in column X=0, and an external 2 lining up with the 3s in column X=1. Note that we keep these external numbers in neat columns! Now the 0 at the far left corresponds to code-23 in the code-column, so everywhere there's a 0 in the original internal columns, we MUST ensure that it gets changed to 23. Similarly, far-left-1's code is unchanged at code-3, so we leave this untouched in 97b's internal columns. Far-left-2 corresponds to code-2 now, so original internal 2s, of which there's only one, gets recorded as 2 in 97b, and finally, far-left-3 corresponds to code-23, so all original internal 3s get changed to 23. This completes the compilation of table 97b.

Having just added 2s all over the place, we must now concentrate on this figure. As before, beginning with column X=0, we ask our standard question, only this time we'll ask it about the 2s in 97b. And what do you know? The 2s in the X=0 column DO have a consistent external figure lining up with them, namely the 3s in the code column. Ho hum! Let's move to column X=1 instead, and repeat our question. The answer has to be "No", because the first and last 2s in this column line up with an external 3, while the lower two 2s line up with an external 2. No consistent line-up at all!!

NOW FOR CODE-1

Nothing for it but to FORCE an alignment, and thereby create 97c, by entering 1s in the code column to line up with the 2s in column X=1. This means that we have to amend our INTERNAL codings too! Only codes 1, 2 and 3 got changed this time, so let's begin with far-left 1 and corresponding code-13, which tells us to change our original internal 1s to 13. Similarly far-left-2 causes original 2s to be replaced with 12, and far-left-3 causes original 3s to become 123. And that's 97c done!

AND FINALLY CODE-0

Now to process the internal 1s. Column X=0 doesn't have any, so there's no problem there, but column X=1 has 1s running all the way from top to bottom. Nowhere is there a SINGLE, consistent, external figure in the code-column to line up with these four 1s, so let's FORCE it once more by putting the next lowest figure (ie, 0) in the code-column (see 97d), and just as before we'll update the table internally. For example, far-left-0, code-023, tells us to replace all original internal 0s with the code 023, and so on for the rest of the table. Final check of 97d coming up! "In column X=0, do our 0s line up consistently?" YES, they do, with the 3s in the code-column. A check of column X=1, of course, shows internal 0s lining up consistently with external 0s. So we've come to the end of the process, having now successfully coded our original table into a multiple assignment form.
Don't relax just yet though! There's more to come before we can draw our multiple-assignment prototype!! We must first convert 97d to what's known as a "terminal connection table", the procedure again being simple and straightforward, thank goodness! See 97e to follow my explanation.

THE TERMINAL CONNECTION TABLE

Here we'll begin with the skeleton of a prototype-table, with just the far-left codes of 97d in the code-column. Next, we're going to be looking from the viewpoint of the code-column of 97d INWARDS to its X-table. Let me elaborate! We'll start with the 0s in 97d's code-column, and ask "What figure(s) in 97d's X=0 column does 0 consistently line up with?" As there are four external 0s, there is NOTHING in column X=0 that lines up with all four, so we enter a "-" in 97e in column X=0 in line with the 0 in ITS code-column. Back to 97d, to ask ourselves "And what does an external-0 line up with consistently in the X=1 column?" Here it lines up with 01, so in 97e in the X=1 column we write 01 to line up with external-0.
Now for the 1s in 97d's code-column. They line up with nothing in X=0, so we enter a "-" in 97e's X=0 column opposite code-1. These external-1s line up CONSISTENTLY only with the 2s in column X=1, so this is what we record in 97e's X=1 column opposite code-1. Similarly, we record the fact that 97d's external-2 lines up with nothing in column X=0, but with 3 in column X=1.

LET'S THINK A LITTLE ABOUT SOME OF THE CODES

Finally we record the fact that 97d's external code-3 lines up consistently with 023 in column X=0, but, BECAUSE WE HAVE AN ENTRY IN COLUMN X=0, in the case of the X=1 column we'll do a little bit of thinking first. First let's observe that although 97d's external-3 lines up consistently with internal-01, the reverse cannot be said to be true, namely, internal-23 doesn't consistently line up with external-3. Under these circumstances, where a line-up only half-complies, we are free to enter 0, 1, 01 or nothing at all in 97e's last remaining slot. Which shall we enter to give us the best advantage? Now's the time to remember an earlier rule about prototypes, which said that if the same entry appears in BOTH X-columns then no contacts are necessary, a straight wire-connection being sufficient. With this rule in mind, our choice is obvious! We'll insert a 0 in this location, to go with the 0 in the X=0 column. 1 is no use to us, as it would only mean an extra contact.
Sometimes we can complete our connection table a little more quickly by filling in all the "-"s first, if there ARE any in the multiple-assignment table developed. For instance, in 97d, there's a "-" in column X=0 opposite code-012, so we could right away (in our connection table) put a "-" in rows 0, 1 and 2 in column X=0, which would leave only one slot to be completed in this column, plus, of course, we'd have to do column X=1. No "-"s in column X=1, so we can't take a short cut there. This method could save some time, and possibly a lot of checking of external codes against internal ones.

DRAWING THE PROTOTYPE CELL

With the terminal connection table all complete, we're ready to draw the prototype at last! This is shown in 97f, the interpretation being the same as before, namely, input-line 0 goes nowhere via a NC-contact, but goes to both terminals 0 and 1 via NO-contacts. And so on for the rest of the table, noting that input-line 3 goes to output-lines 2 and 3 via NC-contacts, but to output-line-0 via a direct connection (as 0 appears in both columns). As far as power input is concerned ... remember it always came in at input-line-0? Our multiple-assignment table in 97d shows that our old code-0 now corresponds to code-023, so we connect power to input-lines 0, 2 and 3 of our first cell. Power-output is ALWAYS taken off at the highest-numbered output-line only, in our case output-line-3.

Observe that our new prototype cell only requires six contacts and ten springs, compared to the original seven contacts and eleven springs. In view of all the work we've just done, you may feel that this isn't much of a saving, but always remember ... this is how much we save PER CELL, so if we had 199 relays in the chain we'd have saved 199 contacts altogether, PLUS the necessary wiring!! Sometimes the work doesn't save anything at all, and we just end up with a DIFFERENT circuit performing the same function as before. Only one thing is reasonably certain - we won't require MORE contacts or springs!
Here's an example which shows how big the saving in contacts can really be!

ITERATIVE NETWORKS - MULTIPLE ASSIGNMENT - EXAMPLE 2

The specs call for a prototype cell to be designed for a circuit of "n" relays which will give a continuous output, UNLESS two consecutive sets are separated by more than one UNOPERATED relay, in which case the power gets cut off.
You're so good at this stuff now, that I don't think a detailed description will be necessary, and you should be able to feel your way through the problem with the aid of the tables of Diagram 98 alone. I will, however, get you started on the first multiple-assignment table of 98c. This is compiled by writing our original codes to the far-left, then, noting in 98a that Z has an entry for all codes 0, 1, 2 and 3, we enter the highest of these numbers (ie, 3) in all locations in 97c where 97a has 0, 1, 2 or 3, whether in the code-column or in the internal table.

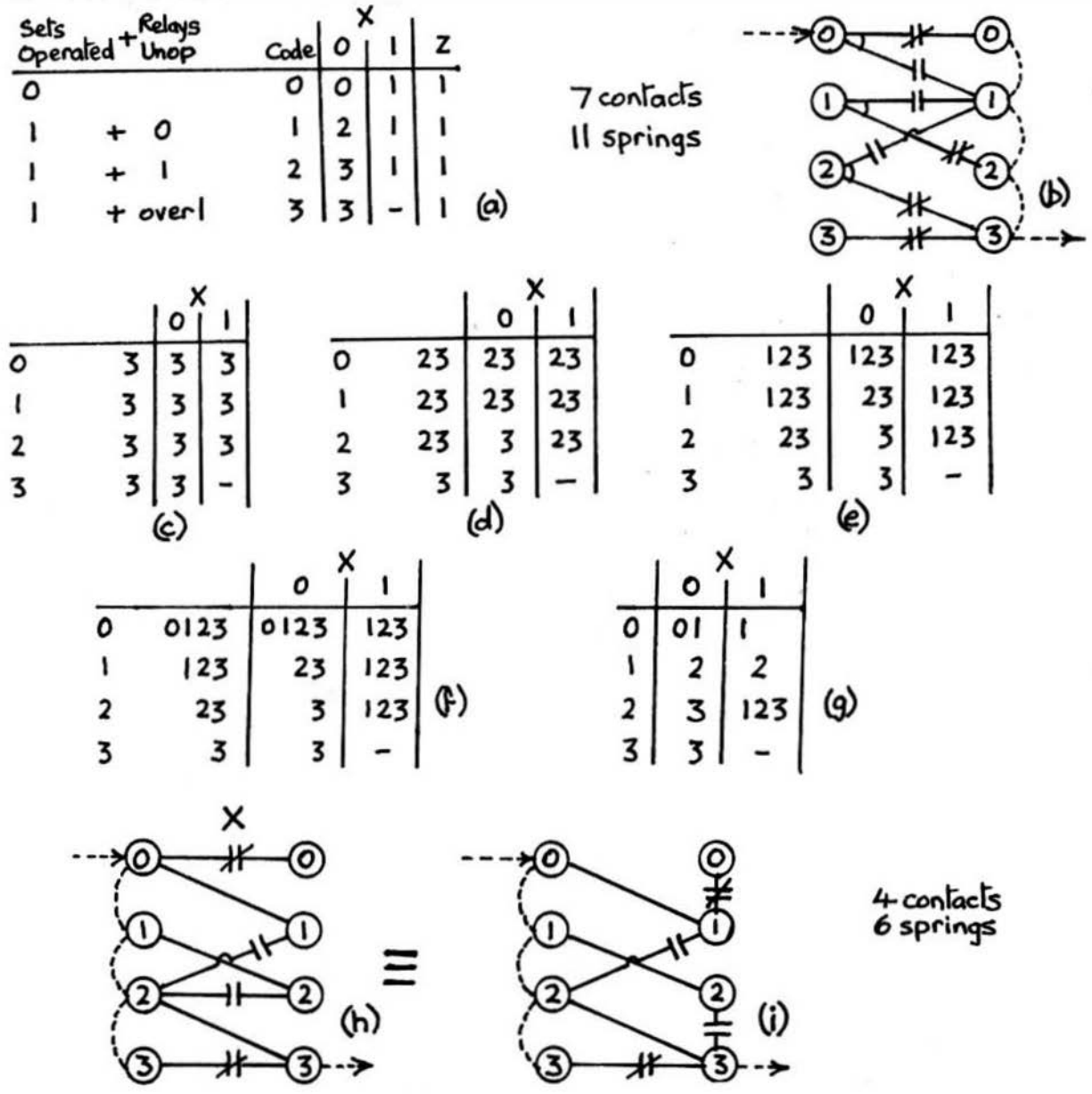


Diagram 98

On second thoughts, maybe I WILL talk you through the construction of Diagram 98g from 98f. First, we'll tackle the "-"s by noting that we have one in 98f, unluckily for us only opposite code-3, so in 98g's code-3, column X=1 we insert a "-". OK, now to 98f's code-column, commencing with 0. This lines up with 01 in the X=0 column, so this is what we record for 98g's code-0 in column X=0. Now, in 97f's X=1 column, 0 lines up with 123, BUT THE REVERSE ISN'T TRUE, as 123 lines up with other rows as well. This means we can pick and choose here, so we'll select only the 1 of 123, to change the 1 already in the X=0 column into a simple wire-connection.
Looking at 98f's external-1 we note that it DOES consistently line up with 2 in column X=0, so let's record this in 98g. Again we have a situation similar to that for our 0-row, namely, code-1 lines up with 123 in column X=1, but not in reverse, so we're free to pick and choose here too. So let's select the 2 of 123 for 98g's column X=1, to go with the 2 in column X=0.
Coming now to 98f's code-2, we note that it, too, only half-complies. That is, although it lines up with 3 in column X=0, internal-3 doesn't consistently line-up with external-2, so let's withhold judgment for a moment while we look at column X=1. No doubt at all here, 2 lines up with 123, which we record in 98g's column X=1. And so, of course, we have no option but to put our 3 in column X=0 in order to neutralise the 3 in column X=1. OK?
Finally, in 98f, external-3s line up with internal-3s in column X=0, and we end up by recording this fact.
When we come to power up our prototype in 98h, we observe that, according to 98f, our original code-0 is now code-0123, so we connect power to input-lines 0, 1 , 2 and 3 of our first prototype cell in the chain, and connect our output-device to output-line 3 of the final cell, this being the highest code-number.

RE-ARRANGING THE CONTACTS IN A PROTOTYPE CELL

Observe how I've shuffled around the contacts of our prototype to form a new cell in 98i. Because input-line-0 is connected DIRECTLY to output-line 1, it doesn't matter in theory whether one end of the upper NC-contact is connected to input-line-0 OR to output-line-1, AS THEY'RE THE SAME POINT ELECTRICALLY. In actual practice though it makes a tremendous difference, as I can now use a single transfer-contact with its centre-point connected to output-line-1. Similarly, I can transfer one end of the NO-contact on input-line-2 so it connects instead to output-line-3. to produce another transfer-contact.
Alternatively, commencing once more with 98h, I could equally as well have shifted the end of the NO-contact from output-line-1 to input-line-0, to form a transfer with its centre on input-line-0. And the end of the NC-contact presently connected to output-line-3 COULD have been shifted to input-line-2. Or any combination of all these shufflings around!
Now for the big surprise! If we count the number of contacts and springs in our new prototype, we find that we've reduced the original 7-contacts/11-springs to a VERY compact 4-contacts/6-springs!! A substantial saving indeed! Especially when you consider that we can now use smaller-sized relays. And only six wires going to each relay instead of the original eleven!! So you see, it IS worthwhile to try multiple-assignment if you're into iterative networks at all.
In fact, it's such an interesting technique, that I'd now like you to do

TEST FOURTEEN-B

Commencing with the regular prototype tables of TEST FOURTEEN-A, convert them to multiple-assignment tables, and compare the original contact/spring rating with the new.

CHIT-CHAT TIME AGAIN

And that, I'm afraid, is just about all I can tell you about iterative networks. This stage of our journey is a fairly critical one, as the next few miles are likely to be a little rough, though I'll TRY to explain things as simply as I can. If you can push your way through that stretch of jungle, you'll not only have a DEEP understanding of some of the most powerful aspects of network design, but will know a LOT better what Boolean Algebra IS and what it ISN'T. However, just in case it all proves a little too much for the lesser-experienced of you, I've arranged for helicopters to lift you over this stretch, and deposit you in a jungle holiday-camp where you can swat mosquitoes, relax, swat mosquitoes, bask in the sunshine, swat mosquitoes, etc., until I and your more foolhardy companions catch up to you again. In other words, you can skip the next few sections, dealing with Boolean matrices (of all things), if you wish! I'd recommend though that you come with us for a little way at least!

... End of Mile 18. At Mile-19 marker, wondering just what sort of creepy-crawlies and other unmentionables lie waiting in the jungle ahead. Shudder! Shudder! Not that I'd want you to lose any sleep over it. Remember the helicopters!

+++

INTERFACING TO THE MOTOROLA 88000 RISC CHIP SET

BY: Terry Lawell & Sang Quan
MOTOROLA Inc.
3501 Ed Bluestein Blvd.
PO Box 6000, Austin, Tx. 78762

MOTOROLA INC.

With the recent introduction of the Motorola 88000 family of high performance RISC architected ICs, a new era in performance oriented microprocessing has begun (refer to figure 1). Like many other performance oriented systems though, the performance of the 88000 family is determined by many factors. These include factors such as the raw processing power of the cpu, the cpu architecture (Von Neumann or Harvard), memory management, code/data caching and the interfacing techniques to the user environment. It seems that in many systems designs interfacing to the user environment does not receive the attention it deserves. In developing the architecture for the 88000 family , Motorola paid great attention to this particular aspect of the system design and developed a new bus interface called the M-Bus. Prior to discussing the details and attributes of the M-Bus, a brief discussion of the 88000 family and philosophy behind the architecture will give the reader a better appreciation for the approach used in the architecture of the M-Bus.

The 88000 family of RISC devices includes the 88100 CPU which provides the integer and floating point processing power for the family and two 88200 CMMUs which provide the memory management and data caching for the data and instruction code. The 88100 uses a Harvard Architecture (totally separate paths for the instruction code and the data) for its external interface and is intended to interface directly to the CMMUs via the P-Bus interface. This interface is optimized for connection to the CMMUs to insure zero wait state access of code and data, but does not preclude the user from connecting the 88100 directly to other memory or I/O devices. When connected to the CMMUs, the 88100 can support up to 4 CMMUs on the code bus and 4 CMMUs on the data bus. The internal architecture of the 88100 is architected for pure performance (refer to figure 2). The internal architecture is organized into four independent execution units with their own separate pipelining capabilities. The 88100 sports a completely synchronous bus interface with separate 32 bit data and address lines and operates at 20mhz. The cpu is capable of executing one instruction(including floating point) every clock cycle while concurrently loading or storing data at a rate of up to one word per clock cycle. With the configuration shown in figure 2 Motorola can guarantee the customer that as cpu clock speeds are scaled upwards, the CMMU speeds will be scaled to match. The 88100 supports many more features than just those mentioned here, but the ones mentioned will give the reader at least a glimpse of what the 88100 is capable of providing.

The 88200 CMMU provides the memory management and caching functions for both the data path and the code path. As mentioned earlier, a typical system would include two CMMUs, one on the data side and one on the code side of the cpu. The organization of the

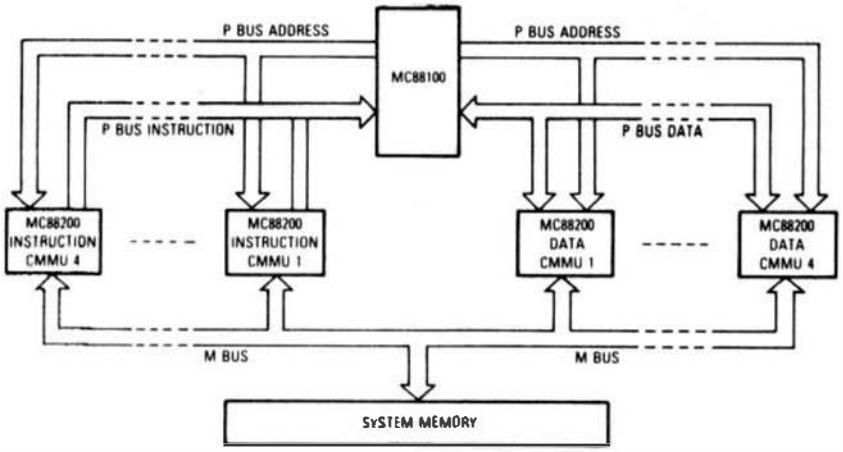
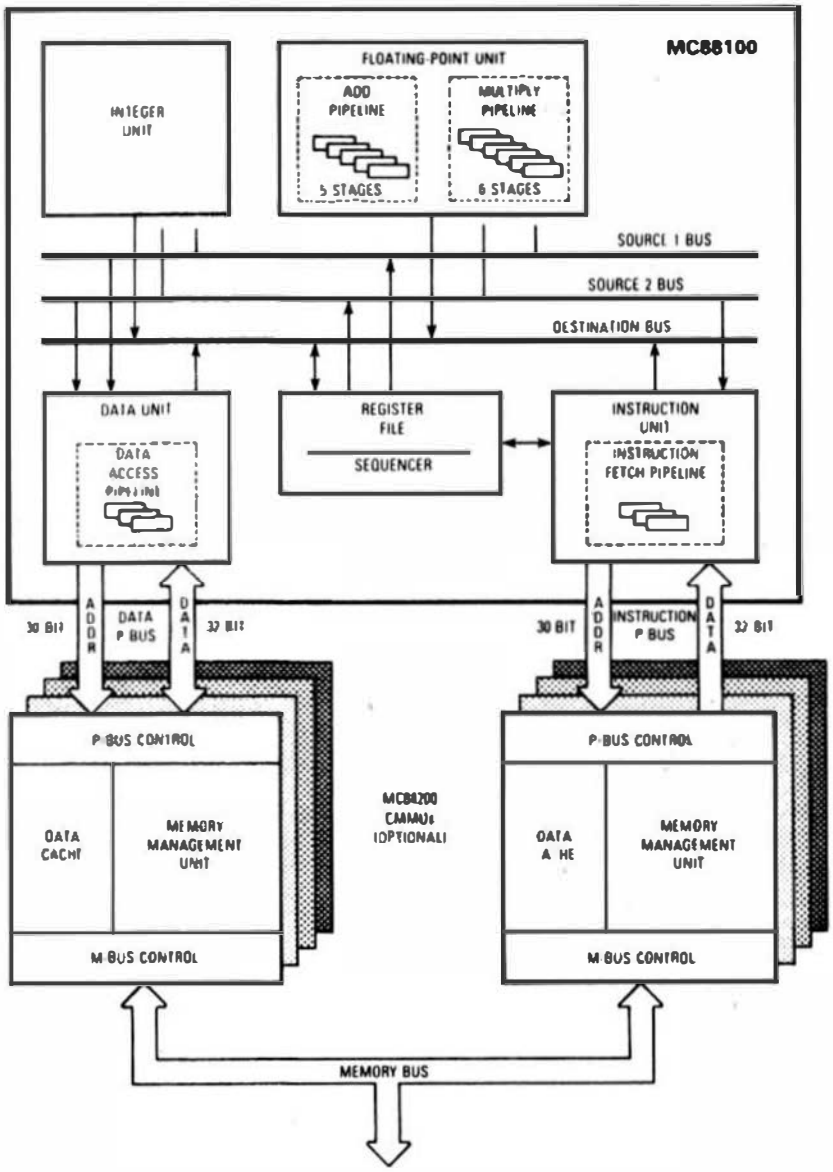


Figure 1 System Block Diagram

Figure 2 88100/88200 Block Diagram



CMMU is quite modular and divided cleanly between the memory management and data caching functions (refer to figure 2). The memory management portion of the CMMU consists of two address translation caches which provide the user with different levels of granularity for address hit detection. The PATC (page address translation cache) provides the user with 56 entries each covering 4k bytes of address space and is updated automatically on an ATC miss by table walks. The BATC (block address translation cache) has 10 entries that provide 512k bytes of address translation each and are loaded and updated by software. The data cache portion of the CMMU, though a totally separate unit from the memory management unit (MMU), works in parallel with the MMU to provide a "no wait state" data caching function. The data cache is organized as a 16-kilobyte four way set associative physical cache. To achieve the "no wait state" capability, the selection of the data cache set is performed at the same time the MMU is doing the logical to physical address translation.

In most environments, the CMMU acts as the users interface to the 88100 cpu via the M-Bus interface. The remainder of this article will discuss the details of the 88200 CMMU and the user interface M-Bus.

CMMU Description

To effectively play the role of an active buffer between the processor and the memory subsystem, the CMMU implements interfaces to the P-Bus on the processor side and M-Bus on the memory side. Logical addresses issued by the processor are presented to the CMMU on the P-Bus, translated (if valid) into physical addresses by the CMMU, which are then used to access the data cache, memory or other devices on the M-Bus. These bus interfaces, together with the data cache, have been designed to keep the user from having to develop a complex interface that is capable of feeding the processor efficiently.

The P-Bus interface supports 33 address lines, 32 bit data lines, and control lines such as Chip Select, 4 Data Byte Enables, Read/Write, Lock, and 2 REPLY lines. The bus protocol is synchronous and pipelined, optimized to supply the processor with either code or data at a peak rate of one word per cycle (80 M bytes/sec at 20MHz). For fault detection purposes, each P-Bus output(actually all outputs in the CMMU) has a comparator circuit that checks the signal on the pin against the one that is fed to its driver. If a discrepancy is detected, the ERR (error detect) signal is asserted one cycle later.

To suppport system fault tolerance, the P-Bus interface can be placed in the checker or shadow mode. In this mode, all output drivers are disabled, allowing a master CMMU to be coupled to one or more checker CMMUs. The checkers have access to the code and data streams and execute concurrently in lock step with the master, verifying every output signal but not driving any. If a checker detects a difference between the output of the master and its own internal signal, it asserts the ERR signal.

The M-Bus interface, described in detail in a later section, supports a synchronous 32 bit bus with multiplexed address and data and control signals to perform bus arbitration, specify Read/Write operations, establish exclusive use of global resources, inhibit external caching, signal bus errors, and provide initialization at Reset. TheM-Bus interface works in conjunction with the cache control logic to automatically maintain cache coherency via an efficient bus snooping protocol. This coherency logic insures that when a device attempts to access a memory location that the cache has a modified copy of, the cache will update memory before allowing the device to complete the access. Cache/memory coherency is a critical issue in multiprocessing systems, or in systems with Direct Memory Access devices.

The CMMU can be configured, controlled, and monitored by software through its register file, which consists of four categories of registers:

1. System interface registers: ID, Command, Status, Address, and Control registers.

The ID Register contains a 7 bit code that uniquely identifies the CMMU when its register file is accessed from the M-Bus. These ID bits are set when the CMMU comes out of reset via 7 external pins that are output-only during normal operation. The CMMU ID may be changed dynamically by writing a new code into this register.

The System Command Register allows software to initiate cache flushes and address probe operations. A probe of a logical address returns the corresponding physical address that is mapped to it, and all the protection/control bits pertaining to that address. Flushes and probes will begin after the appropriate command code is written into this register, either from the P-Bus or M-Bus.

The System Status Register reports M-Bus errors that occur during cache flushes, address probes, and memory updates caused by cache snoop hits (snoop copyback error). This register also reports the results of a probe, showing all the protection and control bits that pertain to the address being probed.

The System Address Register supplies addresses for the probe and cache flush commands, and to select set and word addresses for software accesses of the data cache via the cache diagnostic ports. The physical address of an address probe or of a faulted flush is also returned in this register.

The System Control Register allows software to enable parity checking on M-Bus READs, enable snooping of global data accesses on the M-Bus to maintain cache coherency, and select the priority or fairness protocol when the CMMU arbitrates for the M-Bus.

2. The Local Status and Address Registers are updated by the CMMU when a P-Bus transaction such as ld (load), st (store), or xmem (memory exchange) ends in a fault condition. The status register reports one of the fault codes: invalid segment fault, invalid page fault, supervisor violation, write-protect violation, or bus error. The address register contains the physical address of the location where the fault occurs, except in the case of a write-protect fault where no address will be reported. In this case the faulted address will be available in an exception register in the CPU.

3. Memory management registers: Supervisor and User Area Pointers, and BATC (Block Address Translation Cache) write ports. These registers form the basis of two alternate logical-to-physical address translation processes. Each of the two area pointers can be initialized by software to contain an area descriptor, part of which is the physical address of a segment descriptor table which is accessed during a hierarchical table walk (described in detail later) to perform an address translation. A Translation Enable bit in each of these area pointers, when cleared by software, causes the CMMU to bypass the translation process and treat logical addresses presented on the P-Bus as physical addresses.

Address mapping for blocks of 512K bytes of memory are stored in 8 BATC entries via the corresponding write ports. Each of these entries consists of a 14 bit CAM (Content Addressable Memory) containing a logical block address, a 13 bit RAM containing the corresponding physical block address, various control and protection bits pertaining to the block being mapped, and a Valid bit indicating whether the entry is valid. In addition, two hardwired entries provide direct mapping for "control memory" which occupies the top 1M bytes of the total 4G bytes address range.

4. Cache Diagnostic Ports: 4 cache tag ports, 4 cache data ports, and one cache set status register. These ports allow software to access the various components of the cache as CMMU internal register accesses, mainly for diagnostic purposes.

Memory Management

The address space is divided logically into a supervisor and user space of 4G bytes each. The top 1M bytes of the supervisor space is control space reserved for mapping of control registers of M-Bus peripheral devices. The CMMU uses the S/U (supervisor/user) bit which is part of the logical address to select the appropriate space. These spaces can be mapped to a 4G bytes or smaller physical space, at the block (512K bytes), segment (4M bytes), or page (4K bytes) granularity.

The page mapping is implemented in a hierarchy of area, segment, and page descriptors. Each area and segment descriptor contains the physical address of a 4K byte table pointing to the next level descriptors. In the case of a page descriptor, the address points to a 4k physical memory space.

The descriptors also include protection/control bits for the portion of memory being mapped. Each of the pointers in the two area descriptors points to a segment table which contains up to 1024 valid segment descriptors. Each segment descriptor in turn points to a page table with up to 1024 valid page descriptors. Each page descriptor points to the physical page to which the logical address is mapped. Figure 3 shows the configuration of the three descriptors.

The memory management portion of the CMMU efficiently handles logical-to-physical address mapping described above via two ATCs (Address Translation Caches): a BATC (Block ATC) and a PATC (Page ATC) which functions in parallel. The BATC, which maps blocks of 512K bytes, is optimized for operating system uses, or can be used to map any user space memory that is mapped into contiguous 512 K byte blocks. The PATC consists of 56 entries, each consisting of the S/U bit and upper 20 bits of a logical address stored in CAM, the corresponding 20 bit physical address, and protection/control bits for a page of 4K bytes. These entries are formed and replaced by a hierarchical table walk automatically initiated when an ATC miss occurs. Figure 4 shows the format of the ATC entries.

When a memory access is to be performed (code fetch or data LOAD/STORE; code fetches are the same as data LOADs from the CMMU perspective), the processor presents a logical address to the respective CMMU on the P-Bus address lines. The CMMU latches this address and if translation is enabled, performs simultaneous associative searches in both of the ATCs. In the BATC, the upper 14 bits (S/U and 13 MSB) of the incoming logical address is associatively compared to the logical address field of the 10 entries. If a match occurs in an entry, the corresponding physical block address is issued, concatenated with the remaining lower bits of the logical address to form the complete physical address of a word within that block, which is then used to access memory. In the PATC, a similar associative search is performed using the upper 20 bits of the logical address and the S/U bit, yielding the physical address of a 4K byte page. The lower 12 bits of the logical address are concatenated with the page address to form the physical word address within that page. If there is an address match in both ATCs, the one in the BATC takes precedence.

If there is no match in either of the ATCs, the CMMU initiates a two-level table look up in memory to obtain the physical address as shown in Figure 5.

First, a segment descriptor is fetched from one of the segment tables. The physical address of this descriptor is formed by concatinating the segment table address given in the area pointer selected by the S/U bit and the 10 most significant bits of the logical address. The page descriptor is then fetched at the address given by the concatination of the page table address in the segment descriptor and bits 12-23 of the logical address. Finally the physical address of the word to

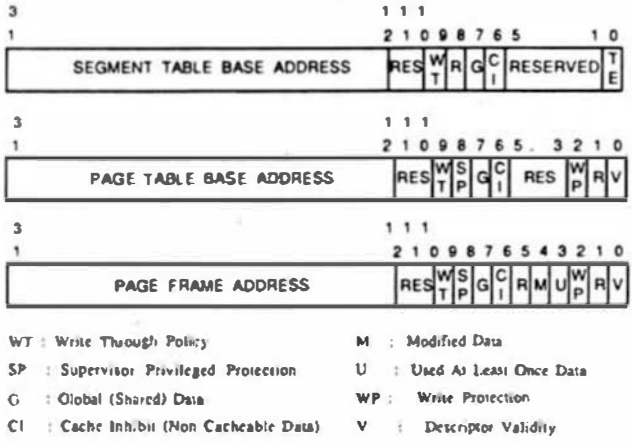


Figure 3 Area, Segment and Page Descriptors

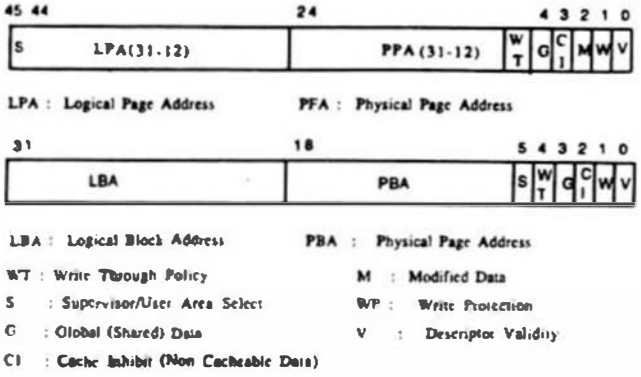


Figure 4 Address Translation Cache Entries

be accessed is formed by concatinating the physical page address in the page descriptor with the 12 LSB of the logical address. Basically, the two groups of 10 MSB of the logical addresses provide offsets into the segment and page tables, and the 12 LSB the offset into the physical page table. The top 20 bits of the physical address and logical address plus the S/U bit, together with all the protection/control bits accumulated from the descriptors are formed into a PATCentry and stored in the PATC, using a FIFO entry replacement policy. The physical address is concurrently presented to the cache or to memory to obtain the word requested in the P-Bus transaction.

DATA CACHE

The CMMU data cache has been designed to exploit temporal and spatial locality, providing 98% or better cache hit rate in most applications. The cache fast SRAM allows the CMMU to supply data to the processor at the sustained rate of one 32 bit word per cycle while there are cache hits.

The 16K bytes of high speed SRAM are organized as 256 sets of 4 lines each, with each line containing 4 32-bit words, with a 20 bit physical address tag, and valid and disable bits as shown in Figure 6. The two valid bits indicate the state of the cache line.

This design, taking advantage of the fact that the lower 12 bits of an address are the same for both the logical and physical address, achieves concurrency between address translations and data cache accesses: the cache is structured so that address bits 11-4 are decoded to select one of 256 sets, while bits 2 and 3 select one of 4 words in the set; this decoding can be initiated and even completed before the MMU has finished the address translation. Once the translation is done, the upper 20 bits of the physical address supplied by the MMU are associatively compared with the 4 physical address tags in the selected set, and a cache hit signaled if a match occurs. The data word is then

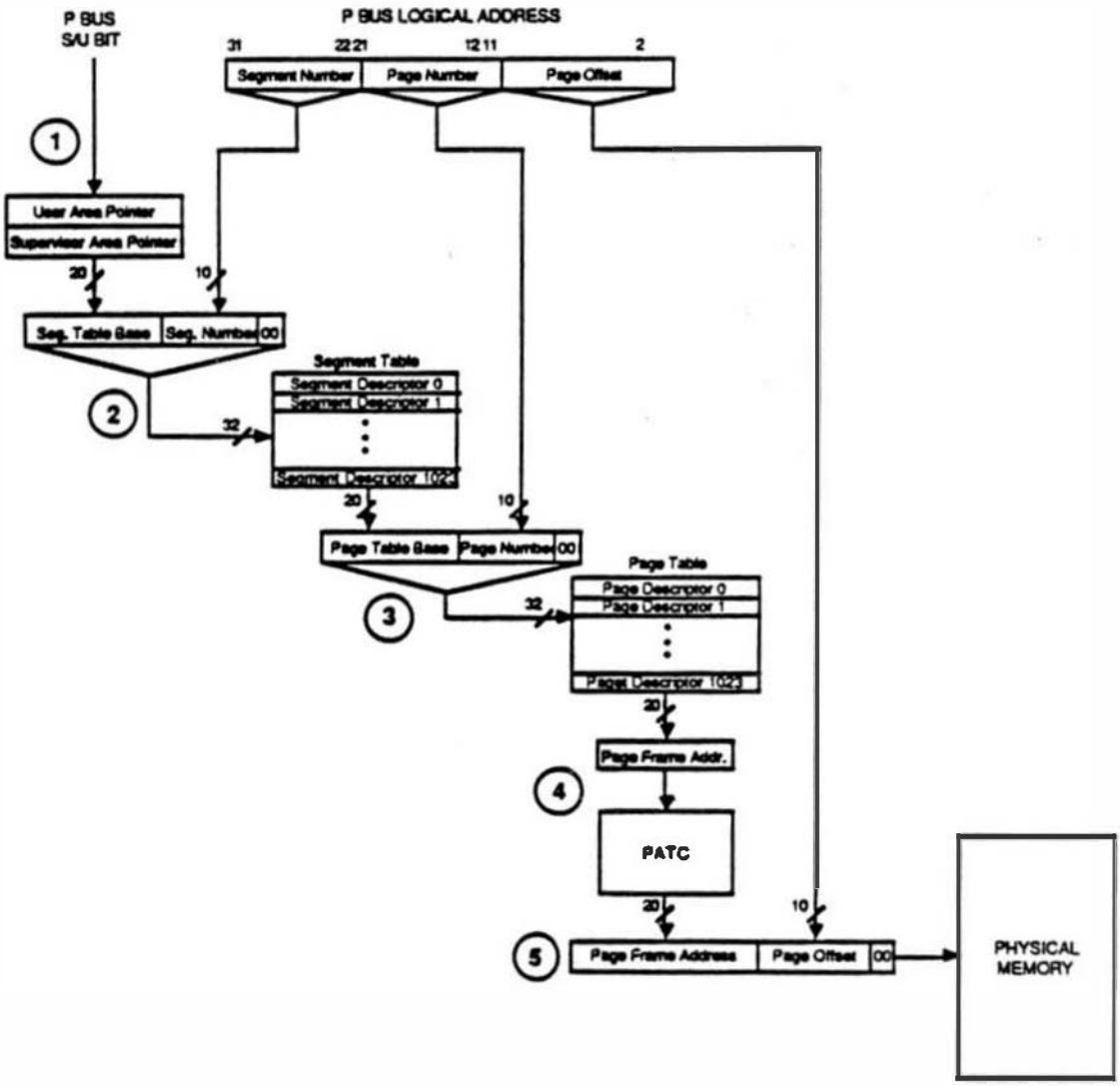
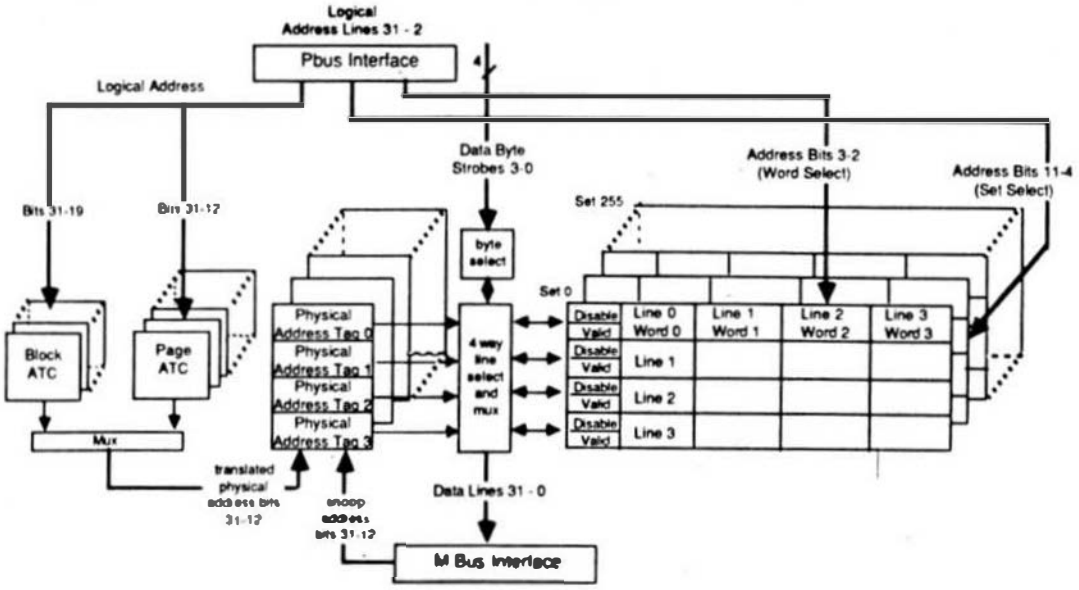


Figure 5 Two Level Table Walk

Figure 6 Cache Organization



supplied to the P-Bus interface to be returned to the processor.

The line structure of the cache, while providing prefetching of code/data, makes best usage of M-Bus burst mode (block transfer) and nibble-mode memories. The number of lines per set also can drastically reduce the probability of replacing a cache line that is going to be accessed in the immediate future.

The cache follows the LRU (Least Recently Used) line replacement rule within a set. It keeps a record of the exact sequential order of line accesses in a set so that the line that is least recently used can be replaced when a cache miss occurs. LRU logic updates the line use order each time a line is accessed or replaced. When a hit occurs, the line that is accessed is marked as most recently used, and the order of the 4 lines in the set is recomputed.

To support fault tolerance, each cache line can be tested and disabled via the disable bit by system software, if found defective.

Cache States And Memory Coherency

The CMMU offers two methods for updating memory; copyback and writethrough. In the writethrough mode, a cache line is written back to memory every time it is modified, keeping memory always consistent with the cache content. The M-Bus traffic can be high when the cache functions in this mode. The copyback policy can be used to maximize memory performance and to reduce bus traffic.

Under this policy, cache writes are written to memory the first time the data in the cache is written (write once); subsequent write to the same location are not written to memory until the cache line is flushed (invalidated or replaced). This delay in updating memory means that the content of memory is not always consistent with that of the cache, making it necessary to have a bus monitoring mechanism that can tell the cache when a bus master tries to access stale global (shared) data in memory. This allows the cache to update memory before allowing other bus masters to try and access the shared data again. The CMMU supports such M-Bus "snooping" capability, to be described in detail in a later section.

The copy back policy requires that only one cache in a multicache system has a modified copy of any global data. The two valid bits pertaining to a line define the four states of a line as follows:

11 _ Invalid: The line contains no meaningful data

10 _ Shared Unmodified: O t h e r caches may have a copy of this line, and the line is not modified with respect to memory.

00 _ Exclusive unmodified: Only this cache has a copy of this line, and the line is not modified with respect to memory.01 _ Exclusive modified: Only this cache has a copy of this line, and the line is modified with respect to memory.

The cache control circuitry updates these valid bits on a cycle by cycle basis as the cache lines are being accessed, making the proper state transitions, influenced by the global and writethrough bits pertaining to the data being cached.

M-Bus,
the user's interface to the
88000 family

The M-Bus is the user's pathway to the 88000 family devices and the 88000 family's pathway to external physical memory (refer to figure 7). The M-Bus is a 32 bit wide synchronous bus that multiplexes address and data on the same 32 bits and operates at the same clock frequency as the 88200. It is designed to support multiple users and has the ability to function as either a bus master or as a bus slave. The M-Bus includes many features which allow efficient operation in such environments as multiprocessor, multiuser and fault tolerant systems. Prior to discussing these features however, it would benefit the reader to have an understanding of the various signals that make up the M-Bus and their functions. The following is a brief description of the M-Bus signals:

AD0-AD31 (M-Bus Address/Data) form the multiplexed address/data bus. The function of these signals depends on the M bus transaction phase (defined by the M-Bus control signals). During the request phase, the MC88200 drives the physical address onto AD31-AD0. During the data phase, AD31-AD0 are the data input/output lines.

ADP0-ADP3 (M-Bus Address/Data Parity) indicate the parity of the M-Bus address/data lines. The MC88200 uses even parity (parity can be disabled by software), checking parity on reads and generating parity for addresses and memory writes. Each parity signal is associated with one byte of the address/data bus.

C0-C6 (M-Bus Control), when the MC88200 is the bus master, define the transaction phase and type of transaction. The C0 signal defines whether the phase is address (address on AD31-AD2) or data (data on AD31-AD0). During the address phase, C1-C5 indicate intent to modify, read/write, M-Bus lock, cache inhibit, and global address. During the data phase, C1-C5 indicates end-of-request, read/write, and selected bytes.

These signals are inputs during M-Bus snooping and when the MC88200 is accessed as a slave device.

CP (M-Bus Control Parity), when the MC88200 is the M-Bus master, indicates the even parity of the M-Bus control signals.

ST0-ST3 (M-Bus Local Status) indicate the local M-Bus status when the MC88200 is being accessed as a slave device (register access), or when the MC88200 is snooping a global M-Bus transaction. These signals are inputs during reset and are used to initialize the CMMU ID register.

SS0*-SS3* (M-Bus System Status), are the reply generated by M-Bus slaves in response to the MC88200 address and data phases.

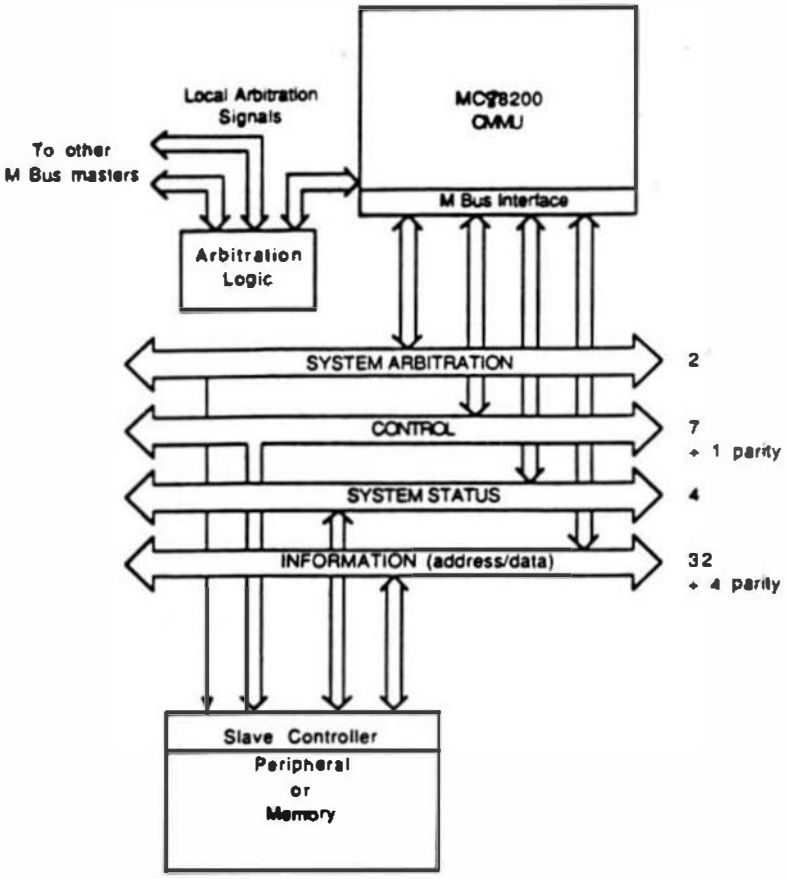


Figure 7 M-Bus Block Diagram

BR (M-Bus Request) is asserted by the MC88200 to request M-Bus ownership.

BG (M-Bus Grant) is generated by external M-Bus arbitration logic in response to a bus request. The MC88200 recognizes this signal only if the M-Bus is not busy (BB* signal negated).

BA (M-Bus Acknowledge) is asserted by the MC88200 when it has received a bus grant in response to a bus request. This signal allows the MC88200 to accept and maintain bus ownership while making memory accesses. It is driven active when BB is negated.

BB* (M-Bus Busy) indicates that some other M-Bus device is currently the bus master, it qualifies the bus grant signal. For an MC88200 to become the bus master, it must receive a bus grant signal from the arbitration logic, and BB* must be negated (no other device is the M-Bus master).

AB* (M-Bus Arbitration Busy) indicates that one or more M-bus devices are performing a bus request. It is an input to the MC88200 that indicates contention for bus ownership is taking place.

MCE (M-Bus Checker Enable) determines the operational mode (master/checker) of the MC88200 M-Bus. In the checker mode of operation, all M-Bus signals are placed in the high-impedance state and all M-Bus outputs are monitored as inputs. The master operates normally. The checker compares its internal results with the results read as inputs. If a mismatch occurs, the checker asserts ERR. ERR can be used to prevent the memory access from corrupting the code, data or i/o system.

MULTIPLE USER OPERATION

Now that the reader has a brief description of the M-Bus signals, it will make it a little easier to follow the discussions concerning the features provided to the user by the M-Bus. The M-Bus operation has four distinct events that take place each cycle; bus arbitration, address selection, data movement and status response. As mentioned earlier, the bus is a multiplexed synchronous bus and all actions on the bus are synchronized to the rising edge of the clock (refer to figure 8). In operations where only one word of information is transferred, it takes two clock cycles to complete the transaction, but transferring information on the bus in 'block mode', only takes n+1 clock cycles to complete(in the case of the CMMUs, 'block mode' transfers four words at a time). The CMMUs perform all cache updates and copyback operations in the block mode. Block mode is performed on quad-word boundaries, i.e. the first word transferred starts at address hex"XXXXXXX0" and ends at hex"XXXXXXXC". At the end of each address and data transfer, including block transfers, all M-Bus participants place their status word onto the bus. This sequence of actions takes place on every cycle that requires access of data. The following discussion will familiarize the reader with the actions that take place within each major operation on the bus.

BUS ACCESS

M-Bus transactions take place between one bus master and one or more bus slaves. Bus mastership is determined through arbitration while slaves are determined by address decode. The arbitration signals offered on the M-Bus allow the system designer the flexibility to implement a variety of schemes as long as the end result is only one bus master being enabled at a time. The five signals supplied for bus arbitration (described earlier in the signal descriptions) are used to access the M-Bus. These signals work in coordination with each other to provide efficient access and departure on the bus. Below (refer to figure 9) is a simple timing diagram of the interaction of these signals:

It is interesting to note that even though the M-Bus allows only one master on the bus at a time, slave devices are allowed the opportunity at the end of each address or data cycle to respond with status conditions that could potentially cause the current bus master to end its tenure and re-arbitrate for the bus at a later time. This

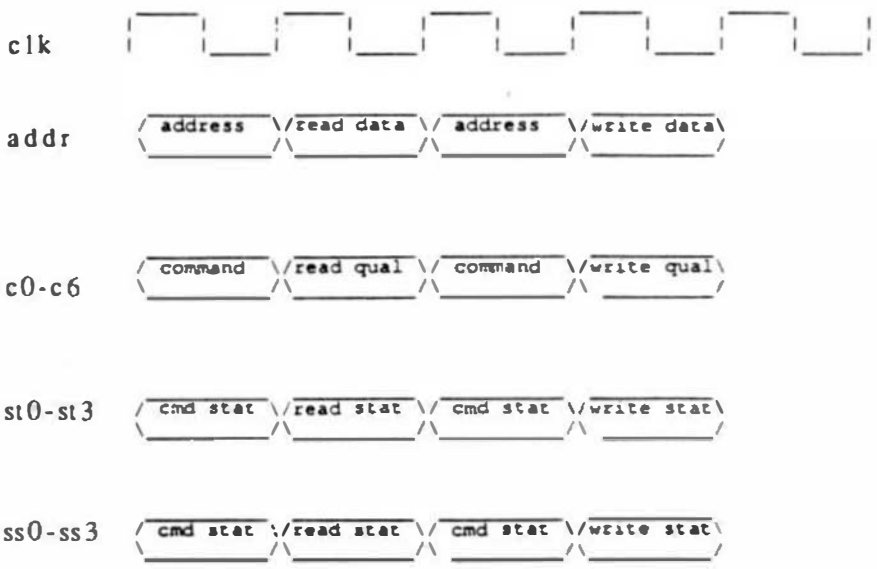


Figure 8 M-Bus General Timing

ability for slaves to impact the bus operation will be discussed a little later. As each potential bus master requires access to the bus they assert BR to the bus arbitration device(implemented externally). AB is the nored output of all BR on the M-Bus (also implemented externally). The arbitration device will then assert BG to the next potential bus master based on its own priority scheme. The device receiving the BG will gain control of the bus as soon as the BB signal is deasserted. BB is deasserted when the current bus master releases its BA. It is possible that during the time gap between receiving BG and BB being deasserted that the bus master elect could lose access to the bus due to a higher priority device being given BG and their BG being removed or that a bus cycle ended in a status of retry or error. In either case the master elect would have to re-arbitrate for the bus. Once a device does receive ownership to the bus they will assert BA. The bus master may retain the bus until other devices require access either for new transactions or to recover from exception conditions. Once the current bus master loses the bus, there is an option in the CMMU that will allow that device to immediately reassert BR (Priority Request) if necessary or wait until AB is deasserted (Fairness protocol) before the BR is asserted.

ADDRESS GENERATION AND DETECTION

The request (address) phase of a bus access is indicated by control bit C0 being asserted . At the end of the request phase, C0 is deasserted and the bus operation enters the data phase. In the master mode of operation, addresses are normally output to select a slave device. In certain conditions however, the address generated may contain the control register address and ID of one of its own internal registers. This is called a PIRA (P-Bus Internal Register Access). In this particular condition, the CMMU acts as both a master and a slave. It acts as a bus master by arbitrating for the bus and generating the address; and acts like a slave by decoding the address. On each single word access, the address is output and a reply is received to either validate that address or to indicate an exception condition. In block transfer

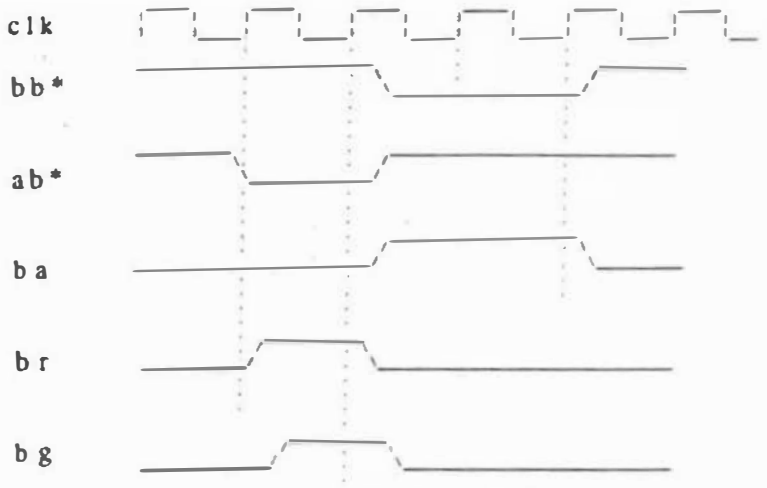


Figure 9 Bus Arbitration Timing

mode the address is generated for the first cycle but not for the last three cycles. It is the responsibility of both the master and the slave to update their respective address counters internally while performing block transfers. The CMMU, while functioning as a slave device, does not use a chip select in the normal sense, but has an ID which is loaded into the CMMU at power-up and can be loaded by software as well. This ID is checked on each request phase to determine if that access is intended for that particular CMMU. The ID has to accompany every address that is intended to access a CMMU from the M-Bus or the P-bus. The request phase is terminated when the OK reply is received via SS0-SS3.

READ AND WRITE ACCESS

Data transfers across the M-Bus are accomplished via reads or writes in either a single or a block mode. In single word transfers the data is preceded by an address and followed by the appropriate status to confirm the operation. If the user intends on transferring less than one word of data, a single word transfer is executed with the appropriate byte strobes enabled. In block mode, the data is transferred in burst of up to four words. A request phase precedes the first word transfer with an address that is byte aligned. Each word transfer is followed by a status condition that qualifies that word. Block transfers are terminated by the master asserting an End of Request (control pin C1) in the data phase of the last word or by the slave asserting an End of Data status for the last word it can handle via SS0-SS3.

STATUS REPLIES

One of the very unique features of the M-Bus operation is the status reply phase. In this phase, all operations are validated, invalidated or redirected depending on the response to that phase by each participating slave device. These status replies are communicated on the bus via the system status lines described earlier. The information conveyed by this status mechanism is how the M-Bus is able to support such features as multiprocessing. The status lines are mutually exclusive with an inherent priority defined through the use of active low level inputs. The status reply with the most active low inputs has highest priority.

Every request and data phase has a corresponding status response presented on the M-Bus at the end of that phase. Since only one master is active at a time, it is the responsibility of each active slave to place the appropriate status on the bus at the end of every phase (refer to figure 8). There are five possible responses that can be placed on the bus by participating slaves; 1) Error 2) Retry 3) Wait 4) End of data 5) OK.

The Error response is normally placed on the bus by a slave device that has detected a parity error, but system implementation allows the developer of M-Bus slaves to apply other error conditions to this response. When an error is detected, the bus master loses ownership of the bus and has to return to the bus arbitration phase.

The next status response, Retry, is the normal mechanism used in multiprocessor systems to handle Bus Snooping(Snooping is described in more detail later). It is also possible for Retry to be used in certain applications as a priority mechanism for higher priority devices to gain access to the bus as soon as it is required.

The Wait response is used to delay the progress of either a request or data phase for various systems reasons. It can be applied by memories with slow access times or by memory controllers who are capable of setting up blocks of memory for burst operations. CMMUs that are Snooping will assert Wait states to do address interrogation. As long as Wait states are asserted by any slave device, the current cycle, whether it is a request or data phase, is repeated on the bus.

End of data (EOD) is used by participating slave devices in the data phase to terminate a burst transfer if the burst cannot be successfully completed. It will also be asserted by CMMUs if a burst is attempted by another device and that burst is not quad-word aligned. In the request phase EOD is decoded as a proper OK response. EOD also allows a slave to respond with a single word when a CMMU requests four words. The CMMU will then request the next three words one at a time.

OK is the normal response to operations that complete successfully, whether they are single word transactions or burst transactions.

BUS SNOOPING

Bus snooping is the unique feature supported by the 88200s to facilitate multiprocessing through the use of shared (global) memory. For those not familiar with the term Bus Snooping, it is the ability of caching devices to monitor the memory accesses of current bus masters and to intervene if the memory location the master is reading or writing is shared memory and has been modified by that caching device, but not updated in memory. When a slave device asserts Retry in response to a memory access, the bus master will immediately relinquish the bus and wait one complete cycle before re-arbitrating for the bus. The one cycle of dead time is to allow the device who asserted Retry to gain control of the bus and update memory. In order for this snooping function to occur, snooping has to be enabled in the CMMU and the memory location being snooped has to be flagged as global. Snooping CMMUs monitor only those bus cycles that are accompanied by the global bit (C5) being set. Each request phase is given an Ok reply by snooping CMMUs. However, the data phase of a global transaction is given two Wait states to afford the snooping CMMUs time to determine if there is a match (snoop hit) in their cache tag. If no match is made, the snooping CMMUs will respond to the next data phase with an OK. If a match is detected and that location is dirty (contains data that is different from that in memory), the detecting snooper will assert Retry on the bus and then BR. The next cycle will be given to the snooper who in turn will update memory with the correct information. If the snooped transaction is a CPU read, the data is copied back to memory, but retained in the cache of the snooping CMMU with different status. If the snooped transaction was a CPU write, the data is copied back to memory and removed from the cache of the snooping CMMU. Snooping is absolutely essential in multiprocessing systems that use the "write back" policy for their respective CMMUs.

FAULT TOLERANT SYSTEMS

There are several features supported on the M-Bus that will benefit fault tolerant systems designs. Features such as output pin comparison, shadowing and parity generation and detection help the system implementer to acheive a more reliable system at a lower cost.

Every output pin on the M-Bus has an internal comparator that compares the output value on the actual pin to the input value to the output driver. Any mismatch will drive the ERR pin active indicating a bus fault.

Shadowing is implemented on the M-Bus by setting a specific input pin (MCE) on the CMMU. When MCE is set, all M-Bus signals are placed in the high impedance state and all outputs are monitored as inputs. The shadowing CMMU monitors all of the transactions from the master and if any mismatch occurs, the ERR pin is asserted.

Parity generation and detection is enabled by setting a control bit in the System Control Register of the CMMU. When enabled, even parity is checked for all read data phases as well as for the control signals that accompany each cycle. Parity is generated on all request phases and write data phases along with parity for the outgoing control lines C0-C6. Any parity errors detected on the M-Bus are reported back to the processor via the System Status Register.

The CMMU also has an internal feature that allows the user to test each line of the cache and to diable any cache lines that test faulty. This provides the user with a mechanism that does not require the storing of a fault map for the cache that has to be referenced on each access.

As can be seen by this brief look into the features and operation of the 88200 and associated M-Bus, Motorola has created a very powerful, yet straight forward approach to handling the many systems concerns that normally surround the design of performance oriented applications. For a more detailed understanding of the features and functions described in this article, Motorola has available Technical Summaries and User's Manuals for the 88100 and 88200 as well as working devices and all necessary software such as compilers, assemblers and functional simulators.

The Macintosh™ Section

Reserved as

A place for your thoughts

And ours.....

Mac-Watch

OF MICE AND MACS . . .

Reviews of Stepping Out II and Quickeys; Information on an upgrade to ImageStudio

By James E. Law
1806 Rock Bluff Rd.
Hixson TN 37343

How Many Mice?

It's always interesting to check out the Macintosh software reviewed in other magazines, especially if it is for a product I have reviewed or which I use. I sometimes find myself saying, "How can they give 5 'mice' (out of a maximum possible of 5) to that dog of a program!" On other occasions it's "That guy is obviously prejudiced against the software. He's picky!" The truth of the matter is that the relationship between a user and his software is a very personal thing and there is room for honest differences of opinion.

Speaking of differences of opinion, why does nearly every program reviewed by some magazines rate a 'superior' rating. Of the 24 programs listed on a single page of ratings in a certain Macintosh magazine, 22 were rated 4 mice or better and 6 programs were rated 5. Is the current batch of Macintosh software that good, or do we just have low expectations? Surely it doesn't have anything to do with advertising dollars? I'm glad to say that the management of Micro 68 Journal has never tried to influence the conclusion of my reviews, even in the case of those few that were quite negative.

A Review of Stepping Out II

The advertising hype for Stepping Out usually says that it appeases ones desire for a large monitor. If so, it would be well worth its modest price. We all get tried of looking at the Mac's little 9" monitor. (I just don't get so tired of it that I want to spend $1500 for a large-screen monitor.) The truth is that Stepping Out's function is to allow you to easily move around a document that is larger than the screen. As the mouse is moved so that the cursor approaches the edge of the screen, the screen image scrolls smoothly to expose more of the document. A secondary function of Stepping Out is to provide for reduced images of your document for page previews and magnified views for detail work. Both views are interactive, that is, you may perform work while the image is reduced or magnified.

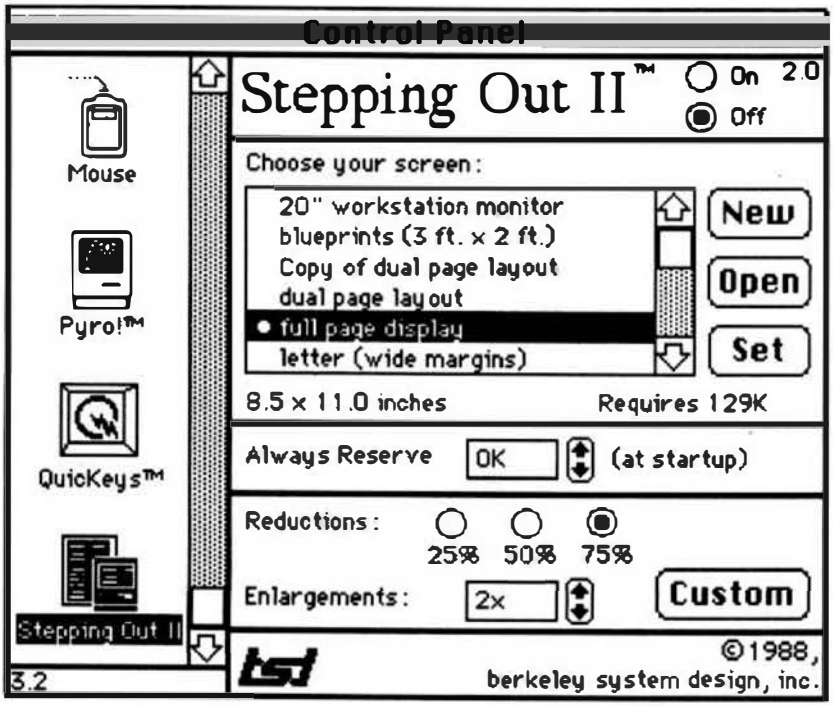
The first version of Stepping Out was generally accepted as being a masterful bit of coding, but some users complained about jerky scrolling and slowed operations. The primary purpose of the latest revision, Stepping Out II, was to improve performance in these areas. To a very large extent, this effort by Berkeley System Design, Inc., was successful. Stepping Out II scrolls quickly and smoothly. While there is still a noticeable reduction in screen refresh time, it was not enough to be objectionable.

Stepping Out II is an INIT, that is, it must be loaded in the system folder and then, upon restarting your Mac, it is ready to use. If you open the control panel you can then turn Stepping Out on and choose the desired parameters. You may choose a variety of document sizes from full size display (i.e., 8-1/2 x 11) to blueprints (i.e., 3 ft x 2 ft). If none of the pre-entered sizes suit you, you may set up a custom size. The control panel also allows you to set parameters for reductions and enlargements and to specify the amount of memory

to be reserved upon start up for Stepping Out II.

Stepping Out II then constructs an "image" in memory of the size page you specified. As the cursor approaches the edge of the monitor, the 'preconstructed' image is fed to the screen so as to provide the perception of smoothly scrolling to show more of your document.

I tried Stepping Out II with page layout programs (PageMaker, ReadySetGo), word processors (MacWrite, WordPerfect), desktop presentation programs (ReadySetShow, PowerPoint), and graphics programs (SuperPaint, Canvas), and very few incompatibilities were identified. The only problem noted was that PowerPoint windows refused to scroll properly under Stepping Out II. Of course, programs with non expandable windows like MacPaint and MacWrite gain little from the use of Stepping Out II (although you can quickly scroll up and down the page in MacWrite).



Stepping Out II Control Panel

If you specify a very large document size, it can be frustrating to work in one corner while the memo bars and tool boxes are at the other corner. Stepping Out II allows you to lock memo bars and tool boxes into place so that they are always visible, no matter where you scroll in the document. The feature does not appear to work with Canvass hiearchical menus that pop out into the area reserved for the document itself.

Stepping Out II makes it easy to magnify portions of the screen up to 16 times its normal size for close up work. The size of the area to be magnified and the degree of magnification is adjustable. This feature works well. The reduction feature does not work so well. Only 25%, 50%, and 75% reductions are allowed. The resulting screens are quite unattractive as the desktop background turns black.

Stepping Out II works with any MacPlus, SE, or Mac II. It is advertised to work with all standard and large Mac II monitors except for 24 bit color. It does not work with many of the big screens used with the MacPlus and SE. Stepping Out is designed to be compatible with all software that works with large screen hardware.

Stepping Out II seems to work, as advertised, with the few problems mentioned above. The question is whether you need extra help in moving around your documents. If scroll bars and built in 'pusher hands' aren't good enough for you, perhaps you need Stepping Out II.

Back to the business of assigning overall ratings. . . Five mice is a lot of mice. I would give Stepping Out II, 4 mice, a solid value; but with room for improvement.

How Quick is Quickeys?

Now I don't want to give up my mouse, but sometimes it's a pain. In the middle of a series of keyboard actions, I have to stop, use the mouse, the return to the keyboard. In the Macintosh Plus manual, there is a picture of a Mac user sitting way back in his chair away from the screen with keyboard in his lap and feet on the desk. We all know that in the real world, this pose wouldn't last for over a few minutes before he would have to re position himself to operate the mouse. Wouldn't it be nice if some of those mouse operations could be handled from the keyboard?

Well, they can with Quickeys from CE Software. Mouse actions such as choosing menu items, clicking dialog boxes and scrolling the screen can be assigned to a key or combination of keys on the keyboard. A single keyboard stroke can also be programmed to perform other actions such as entering a passage of text.

Quickeys, is an INIT and is activated by placing it in your system folder and restarting your Mac. You then, at any time, have access to the Quickeys control box by typing OPTION-CONTROL-RETURN. The combination of keys, like all other Quickey preestablished by key combinations, may be easily changed to suit the individual user.

Quickey entries are made by selecting one of 11 options under the "Define" menu. A very brief overview of the possibilities are as follows:

1. TEXT — Assign a string of text to be displayed when a key is typed.

2. FILE — Close the currently open file and open another application or document without going through Finder.

3. MENU/DA — Open any DA or make any menu selection with a key stroke.

4. ALIAS — The typing of a specific key will cause a different character to be displayed. Use this feature to reorganize your keyboard.

5. CLICKS — Use a keyboard command to replace any mouse click and dragging operation.

6. SEQUENCES — String up to 30 Quickeys entries together into a powerful macro.

7. Buttons-Use a keyboard command to click any button.

8. MOUSIES — Use keyboard commands for specific actions such as zoom or close window, page up or down, or line up or down.

9. SPECIALS — Use keyboard commands for miscellaneous actions such as shut down, restart, or select rear window.

10. DATE/TIME — The designated key combination makes the current date and/or time appear in one of a number of formats.

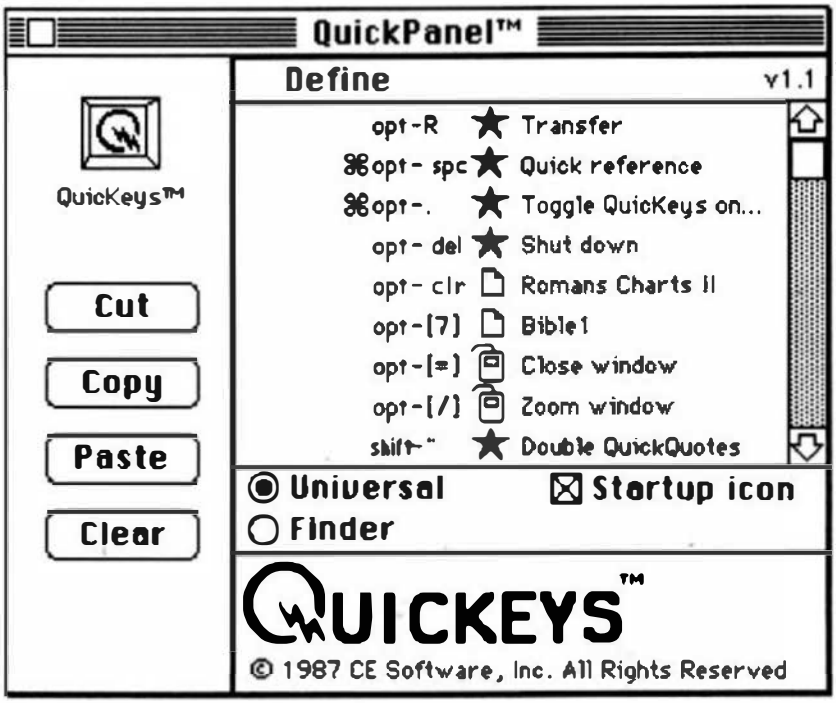
11. F KEYS — Select an FKEY with the designated key command.

Each Quickey key stroke can be assigned to a "universal set" (it is always available for all applications) or may be assigned to a specific application. In cases of conflicts, the key strokes in the program specific sets take precedence over those in the universal set.

To illustrate the usefulness of Quickeys, let me tell you how I have made use of it to speed up my preparation of viewgraphs with PowerPoint. First, I press OPTION-CLEAR to go directly to PowerPoint without going through the Finder. OPTION and "+" or "-" lets me flip through my slides from the keyboard. With OPTION-Z, I zoom back to review the entire slide or with OPTION-A I can expand the slide to full size. OPTION + (arrow key) allows me to quickly page up or down, right or left. OPTION-COMMAND - (arrow key) lets me move more precisely a line at a time. Finally, when I press OPTION-RETURN, my Mac immediately shuts down (after giving me a chance to save) without going through the Finder.

Since I assign the key strokes, I can select something that makes sense to me. It's easier to remember that way. But if I forget, I can enter OPTION-COMMAND-SPACE BAR and a list of all my Quickey entries are displayed.

Quickeys is a handy program. Even with the standard Macintosh keyboard, it is a real time saver. Power users and experienced typists will appreciate not having to take their hands off the keyboard to reach for the mouse. I would have to rate it at least 4-1/2 mice if I were giving ratings based on mice points.



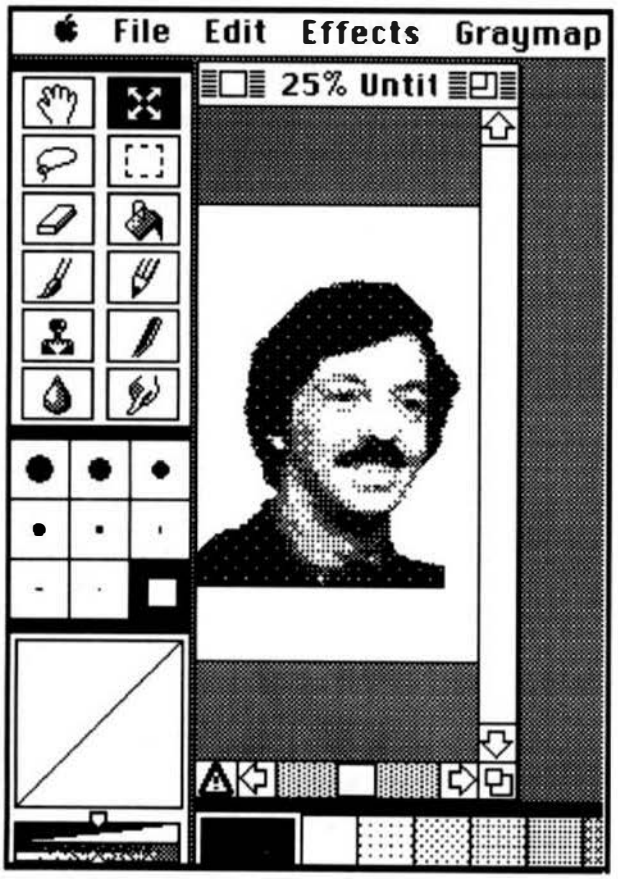
Quickeys Control Panel

Image Studio is Upgraded

Version 1.0 of Image Studio software from Letrasel USA, was named as product of the year by Personal Publishing magazine and now an ever more powerful version (1.5) is available. Image Studio allows you to produce high quality camera-ready halftones on the Macintosh and incorporate them into desktop published documents. The principle improvements in version 1.5 are as follows.

1. Enhanced Grayscale Capability — ImageStudio 1.0 could process up to 64 shades of grey but with Version 1.5, you can manipulate 256 shades. Obviously the number of shades in a file depends on the capabilities of the scanner used. But once you have the image in ImageStudio, you can take advantage of all 256 shades to fine tune the image.

If you have a Macintosh II with an 8-bit video driver and a high resolution monitor, you can view 102 shades on screen. With the Macintosh Plus and SE, you can view 64 shades plus white on screen.



Partial ImageStudio 1.5 Window

2. Memory Management — Version 1.5 provides far more sophisticated memory management techniques which make it possible to work with images larger than your RAM would normally allow. This is important because scanning with 256 shades and 300 dpi resolution creates very large files. ImageStudio handles this feat by creating a working copy of the image on dish. You can then open a portion of an image at a time while you edit.

3. Scanner Control — You can now drive scanners and image grabbers without leaving ImageStudio. The "File" menu now includes "scanner setup" to set options and a "scan" command which activates the scanner or image grabber. This arrangement allows you to scan directly into RIFF format which has the same resolution potential as TIFF but with smaller file sizes.

4. **Upgrade of Tools** — Finally, a number of ImageStudio's tools and commands have been enhanced. For example, you can now scroll selections past the window boundaries and can constrain selections to squares or straight lines.

Version 1.5 of ImageStudio greatly enhances an already powerful program. Those of you who have occasion to use scanned images should check it out.

Please Feed Me Back Some Feedback

I would like to hear from you on what you would like to see in this department of 68 Micro Journal. Would you rather see a number of shorter reviews or a few more in depth reviews? Is there any particular type of software you would like to read about? Let me hear from you.

EOF

Writing Position Independent and Reentrant Code for the MC68000 Family

MOTOROLA INC.

3501 Ed Bluestein Boulevard
P.O. Box 6000, Austin, Texas 78762

By Truman T. Van Sickle

Background

A multitasking system is an computer operating system that will allow the seeming concurrent execution of severaldistinct tasks within a single computer. The several tasksmight be completely independent, or mutually dependent in thesense that calculations or measurements made by one task areneeded as inputs to another. For example in an industrialcontrol system, one task could be used to set several digitaland analog outputs to control a machine. A separate butcommunicating task could monitor sense switches and analogsignals from the machine. The computer inputs indicate thestate of the machine at any time and dictate the next operation that the machine must execute. The machine statemight be stored in an area of memory that is accessible byeither task. With these two tasks running concurrently, theprogrammer will have a table with the current state of themachine. It is not needed to scan the input status as isdone with programmable controllers or single tasking machinesto determine the current machine status. The job of theprogrammer is easier in this case with a multitasking systemthan with a single tasking system.

Some computers employ memory management. A memory manager isa hardware feature that examines the logical address of eachcomputer transaction and determines the correct physicaladdress for this activity. When a programmer writes aprogram, it is usually started at an address zero. When thisprogram is executed, this beginning address must be changedif for no other reason than the MC68000 family of partsdedicate the first 1024 bytes of memory to an exceptionvector table. When the computer has no memory manager, it isrequired that the programmer allocate memory for the variousmemory resident tasks. In this case, it might be necessaryto place the code to be executed anywhere in memory wherethere is sufficient unallocated memory to hold the task.Code that can execute when placed anywhere in memory iscalled position independent.

Reenerant code is a feature that will allow more than onetask to appear to execute a routine at the same time. Theconcern with reentrant code is that the machine and memorystatus associated with the code be secure enough that theroutine can be arbitrarily stopped at any point of itsexecution and another task can start executing the same codewith no interference between the tasks calling the routine.

Features like reentrancy and position independence were notreally considered necessary for microprocessors until theMotorola MC68000 family of parts became available. The mainneed for reentrancy comes about when multitasking operatingsystems are used. Position independence is desirable formany programs in a multitasking system, especially amultitasking system with no hardware memory management. Thepurpose of this paper is to describe the need for bothposition independent and reentrant code, and to demonstratehow easily these important features are implemented with theMC68000 family architecture.

Reentrant Code

Reentrant code means that several different tasks can call areentrant module, seemingly at the same time, withoutinterference. Suppose that a multitasking system isoperating, and at the current instant, a specific routine isbeing executed. Let us use a program like a software-floating point multiply as an example. Task A is in theprocess of computing the product of two numbers. At anytime, a clock interrupt can occur and remove control of theprocessor from Task A to the task dispatcher of the operatingsystem. When such an operation occurs, the task dispatchercan, and usually will, change the running task from Task A toanother task that is on the ready list. Let us assume thatTask B is placed in control of the processor. This processinvolves first saving the status of the computer for Task Aand then restoring the previously saved status of the machineat the last time Task B was in control. Task B is nowrunning.

To make the problem more interesting, let us suppose that thesoftware floating point multiply is a globally shareableroutine. Globally shareable means that anytime a softwaremultiply is needed, the same routine will be used by anytask. Only one copy of the multiply routine will be storedin memory. Everything is okay until Task B needs to executea software multiply. At this time, a routine must be enteredthat was being executed by Task A when Task B took control ofthe processor. There can be no special provisions at theexit of Task A control to save the status of the routinebeing executed. The normal exit procedure at task switch isto save the machine status. Therefore, the status of theexecuting reentrant routine must be contained completelywithin the machine status at all times.

For the most part, the machine status contains the programstatus. The Data and Address Registers are all saved as partof the machine status as is the content of the Condition CodeRegister and the Program Counter. The only missingcomponents of the program status are the values contained indata storage needed for the execution of the routine.

A simple way to avoid storage problems is to store all datain the memory space of the calling program rather than tohave specified data space in the executing routine. Thesedata are all stored on the stack of the calling program, andwhen the stack pointer is saved as part of the normal machinestatus, all volatile data associated with the executingroutine is automatically saved in memory associated with thecalling program. When Task B starts to execute the floatingpoint multiply, any necessary volatile memory will beassigned in the Task B memory space, and the data saved whileTask A was being executed will be safely saved in the memoryspace of Task A. This operation can be repeated indefinitelyand the number of tasks that can simultaneously access agiven reentrant routine is unlimited.

Another application of reentrant code is to implementrecursion in a routine. A function is said to be recursivewhen it calls itself. If a routine is reentrant, it makes nodifference whether the routine calls itself or it isreentered by another program after a task switch. In eithercase the handling of local storage must allow the routine tobe entered while it is set up and executing a call fromanother program. Any recursive routine must be reentrant.

Suppose that a multiuser operating system is being used byseveral programmers. Much of the time that programmers sitat the keyboard they are either editing or compiling orassembling a program. With a well designed operating systemand properly designed system utilities such as editors andcompilers, there is no need for multiple copies of theseroutines to be resident in memory. Assemblers, editors,compilers, and other frequently used programs are each madereentrant. The program loader can examine the task list whena new program is loaded and make use of a copy of the programthat already resides in memory. This approach can provide asignificant memory savings on a large multiuser system.

Writing Reentrant Code

Most programming is done in high level languages. Usuallythe language takes care of providing reentrant code. All Ccompilers and most Pascal compilers can provide reentrantcode. These compilers create reentrant code so long as thereare no static variables associated with the routines. Allvariables should be dynamic. Dynamic variables are stored onthe stack which is passed from the calling program. Staticvariables must be stored locally and associated uniquely withthe local routine. Therefore, static variables will not besaved automatically when the task dispatcher passes controlfrom one task to another, and there is no guarantee that asecond task can enter the routine without disturbing thestatus of the first.

Static variables can be used if they are not used forexecution of the routine. For example, a routine mightrequire initialization the first time it is entered. Here, astatic variable could be used as a flag to indicate that theinitialization has been completed. This variable is not apart of the computation executed by the routine whenever theit is called. Generally, static variables must be used withextreme care with reentrant code.

The discussion here is aimed at assembly language programmingonly. Most of the time, high level languages will sufficefor the every day programming. The times that the high levellanguages fail are those when use of and access to machinelevel operations are necessary. For example, I/O drivers,screen access programs, graphic drivers, and global functionssuch as floating point operations would probably be writtenin assembly language. Both reentrancy and position-independence are required for these types of routines.

The MC68000 family parts have architectural structures andinstructions that make the programming of reentrant codeparticularly easy. The indirect addressing modes allowsaddressing of bytes, words or long words — 8 bit, 16 bit, or32 bit quantities — that are offset from a frame pointer orfrom the stack pointer.

A most convenient instruction is the LINK instruction. Theformat of the link instruction is

LINK An,#displacement

In response to the link instruction, the MC68000 pushes thecontent of the designated address register onto the stack.Then the new value in the stack pointer after the pushoperation is placed into the designated address register.Finally, the value displacement is added to the stackpointer. In the MC68000 family, the stack pointer is alwaysthe address register A7. After execution of the LINKoperation, there is a designated amount of memory on the userstack that can be used as storage by the currently executingroutine. Boundaries of the storage are marked by thecontents of the stack pointer and the contents of An.

The diagrams in Figure 1 shows the impact of the LINKinstruction. In this case, the designated address registeris A6, and the displacement is -14. The memory entries shownin Figure 1 are 16 bit words. Assume that the program has

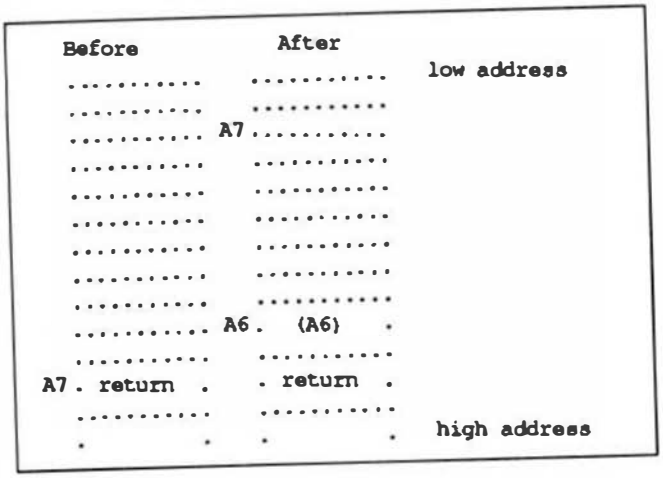


Figure 1. Effect of execution of the instruction LINK A6,#-14

just entered a subroutine. Therefore, the stack pointer ispointing at a location in memory that contains the long wordaddress of the next instruction in the calling program. Thislocation is designated by return in the figure. After theLINK instruction, the content of the register A6 has beenpushed onto the stack, and the new value of the stack pointerhas been placed in the register A6. Finally, thedisplacement -14 has been added to the content of A7 so thatthe final memory configuration is as shown in the figure.

At the end of the subroutine execution, the original statusis restored by execution of the instruction

UNLK An

This instruction causes the contents of An to be placed inthe stack pointer, and the top of the stack is pulled intothe register An. After this operation is completed, thememory status is restored to precisely that shown in theBefore portion of Figure 1.

After execution of the LINK instruction shown in Figure 1,there are 14 bytes of memory allocated to the routine. These14 bytes will remain attached to the routine until an UNLKinstruction is executed prior to return to the callingprogram. After execution of the UNLK, the memory assigned tothe routine is gone. Thus, the name dynamic. The amount ofstorage assigned to each routine is just the amount needed toexecute the routine, and the storage exists only while theroutine is being executed. Also, it should be noted that inthe event that the routine is interrupted by the taskdispatcher, all registers will be saved prior to the startingup of another task. Therefore, when this routine isrestarted by the task dispatcher at a later time , allregisters will be restored and any dynamic memory used bythis routine will be unaltered in the event that another taskmight use this same routine.

The register An used with the LINK instruction is called theframe pointer. Mnemonic names may be assigned to the memorylocations on the stack by any of several means. Probably theeasiest technique is to use the OFFSET directive in anassembly language program. An OFFSET directive allowsdefinition of a table of offsets created by Define Storagedirectives. Symbols thus defined are kept internally by theassembler. The offset table may contain no executableinstructions. The offset table is terminated by a SECTION orORG directive.

As an example, suppose that our routine needs three long wordvariables, ABLE, BAKER, and DOG, and an integer CHARLIE. Thefollowing OFFSET table could be used to create thesevariables:

```
        OFFSET 0

ABLE    DS.L  1
BAKER   DS.L  1
CHARLIE DS.W  1
DOG     DS.L  1

        SECTION 1
```

Here the table is terminated by entry into a SECTION 1portion of the code. When this table is assembled, the valuefor ABLE will be 0 because the offset is 0. BAKER andCHARLIE will be 4 and 8 respectively. DOG will be assigned avalue of 10. Reference to these mnemonics relative to thestack pointer will automatically access the correct memorylocation on the stack. An example that accesses theselocations is as follows:

```
SUB  LINK A6,#-14          Link the stack space
MOVE.L  D0,ABLE(A7)        Place D0 into ABLE
MOVE.W  D0,CHARLIE(A7)     Place the lower word of D0
                           in CHARLIE
SUB.L   D0,D1              Subtract D0 contents
                           from those of D1
MOVE.L  D1,DOG(A7)         And save the result in DOG

etc.
.
.
UNLK A6                    Unlink the stack space
RTS
```

Here all memory operations are relative to the stack pointer.Figure 2 shows how the memory is organized on the stack.

Sometimes the stack is used for other operations during theexecution of the routine. In such a case, the stack pointermight be altered during the execution of the program, and thereferences into the offset table would be shifted. Here allof the memory values would be lost unless extreme care wereexecuted in creating the code. An alternative means existsthat allows memory references relative to the FRAME POINTERwhich should remain unaltered during execution of theroutine. Note the following minor change in the OFFSETtable.

```
OFFSET -14

ABLE      DS.L   1
BAKER     DS.L   1
CHARLIE   DS.W   1
DOG       DS.L   1

SECTION 1
```

This symbol table starts with an offset of -14 rather than 0as was used earlier. Here, ABLE will be assigned a value of-14, BAKER a value of -10, CHARLIE will be -6 and DOG willequal -4. Recall that A6 is pointing to the top of the framepointer. Therefore, offsets generated by this table relativeto A6 will chose memory locations for the various labelswithin the stack area attached by the link command. Thelisting below performs the same functions as the exampleprogram above.

```
SUB  LINK    A6,#-14
     MOVE.L  D0,ABLE(A6)
     MOVE.W  D0,CHARLIE(A6)
     SUB     D0,D1
     MOVE.L  D1,DOG(A6)

     Etc.
     .
     .
     UNLK    A6

     RTS
```

The difference in this case is that all labels are offsetfrom the frame pointer A6 rather than the stack pointer A7.In most cases, either approach works equally well and thechoice of which reference technique depends only on theprogrammers whim.

```
                    Lower Address

                    .
                                        .
                    ....................
ABLE                ....................   A7
BAKER               ....................
CHARLIE             ....................
DOG                 ....................
FRAME POINT         ....................   A6
RETURN              ....................
                    .
                                        .
```

Figure 2
Memory organization for the above example.
Note that CHARLIE is a two byte or integer value.

POSITION INDEPENDENCE

Position independent code can be placed anywhere in memoryand will work with no difficulties. There are severalreasons to want position independent code. Most codeintended to be placed in ROM should be position independent. Therefore, the ROM beginning address need not be kept at aspecific memory location. Another, and probably more-compelling reason, has to do with memory assignment onmultitasking systems where there is no memory manager. If asystem has no memory manager, there are two frequently usedapproached to the assigning of memory space to a program.The first requires that the programmer link the severalprograms to be loaded at unique addresses so that none of theprograms can overlap. This approach really makes theprogrammer into a makeshift memory manager. When it isrequired to load this code, the loader must not allow thememory space of this program to overlap the memory space ofany other running program. A somewhat better approachrequires that all code be position independent. When aposition independent module is loaded, the loader must merelyfind free memory space large enough to contain the programand load it there. Once the program is started, it willoperate exactly the same as if it had been assigned to adesignated memory space. Since the system is multitasking,there will be several modules loaded into the memory space ata time, and the operating system has the responsibility ofmaking certain that there is no memory interference betweenthe modules. Otherwise this job is faced at link time by aprogramer who might not even know what modules will be loadedinto the computer at any given time.

Writing Position Independent Code

A provision in many assemblers allows grouping blocks of codeor data into sections. Motorola assemblers for the MC68000family of parts provide for sixteen sections for each module.Section organizations allow the programmer to specify specialcharacteristics that need to be grouped when the program islinked into a working module. For example, dynamic ramstorage could be placed on one section and high speed staticram in another. Utility programs stored in ROM could be in asection, and applications programs could be linked into yetanother section.

When a program is assembled, a symbol table is created thatcontains the relative address of every label used in theprogram. These addresses are relative to Section starts withthe MC68000 family assemblers. If there are several sectionswithin a given program, each section start address is set tozero. Therefore, the symbol table must contain both therelative section address of each label as well as the sectionin which it is contained. Code generated by an assembler isof the relocatable object format.

Another parameter kept by the assembler is the current-location counter. This value can be accessed by theprogrammer by use of an asterisk in the operand field. Forexample, a command to load the current location would be

```
LEA   *,A0
```

This instruction will cause the current value of the locationcounter to be loaded into the address register A0.

One or more relocatable object modules are combined into asingle loadable object module by the linker. The linkerpulls all of the like sections together and assigns them tomemory locations. After linking, the program instructionaddresses, earlier called location counter values, will becalled program counter values. When a program is loaded intomemory and the program counter

is given the value of thefirst instruction in the program, the program begins toexecute. The instruction pointed to by the program counteris fetched into the machine to be executed, and the programcounter value for the next instruction is calculated beforecompletion of the first instruction. This sequence isrepeated progressively until the program is completed.

It is not necessary to place the various sections of aprogram in contiguous memory. In fact, if the sections areused as outlined above to designate special memory character-istics, it would be impractical to always expect the datafrom any program to occupy contiguous memory. Therefore, thelocation of data storage in a computer may not ever bepoisiton independent.

The above discussion might lead us to believe that theprogram, after linking, must be placed at some designatedlocation in memory and remain there. Seemingly, dataaddresses must be fixed in some way so that they can befound when different portions of the executing programrequire them. There is a way around the problem. MC68000instructions can use the difference of the data address andthe program counter to locate a data address. Such anaddressing mode is called Program Counter Relative, PCR. Ifall data accesses and branches — including branches tosubroutines — are PCR, there is no need for absoluteaddresses anywhere in the program. If no absolute addressesare needed, then the program can be executed anywhere inmemory, and it is indeed position independent.

To create position independent code, the programmer must makeall data accesses and changes in program flow, JMP, JSR, BRA,Bcc, and BSR program counter relative. Any branch instruction is automatically PCR. The only concern with a change inprogram flow is to make certain that the branch instructioncan reach the destination. This concern will be discussed atgreater length later.

For data accesses, the PCR mode can be used. Function codesof the MC68000 parts contain hardware signals of a dataaccess or a program access. Any storage operation that isPCR implies that the operation is both program and data atthe same time. A PCR operation causes access of the programcounter which will assert Program Function Code line. Thefact that anything is being stored in memory will cause theData Function Code line to be asserted. These two linescannot be asserted at the same time. Therefore, this type ofoperation is not allowed in any of the MC68000 family ofparts. The addressing category that designates an effectiveaddress in which the data can be stored is called DataAlterable. Data Alterable destination effective addressesfor a PCR operation are specifically not allowed. Suddenly,the problem becomes a little more complicated.

Most arithmetic and logic instructions are of the form

```
OPCODE   <ea,Dn
```

or

```
OPCODE   Dn,<ea
```

where <ea is the effective address, and Dn is a datareg-ister. The left most operand is the source operand, andthe right is the destination operand. The effective addressadmits a total of twelve different addressing modes. As wehave already seen, not all instructions will permit all ofthe different addressing modes as either source ordestination operands. If the assembler option

```
OPT PCS
```

is included at the beginning of a section, all code assembledin that section will automatically be assembled as PCR typeoperations. Therefore, the code

```
ADD    ABLE,D0
```

will be assembled as

```
ADD    ABLE(PC),D0
```

However, the code

```
ADD    D0,ABLE
```

will be assembled exactly as written because the PCR modecannot be used for Data Alterable operands. As a result, itis the responsibility of the programmer to replace this codewith a sequence like

```
LEA    ABLE,A0
ADD    D0,(A0)
```

to achieve a position independent equivalent to the abovestatement. The Load Effective Address, LEA, admits all PCrelative operations. Therefore, this sequence will providefor a Data Alterable operation that is PC relative.

In the operand description

```
label(PC)
```

the item label is called a displacement. The effectiveaddress is calculated as the sum of the current ProgramCounter Value and the sign extended sixteen bit value label.All parts of the MC68000 family have a 16 bit limitation tothe size of the displacement in this case. For the MC68000and the MC68010 parts, this limitation exists for the branchinstructions as well as the displacements. If it becomesnecessary to extend operation beyond the plus or minus 32kbyte range dictated by this limitation, other approaches canbe used.

Another PC relative addressing operation is the indexed mode.This address mode is described as follows

```
label(PC,In)
```

where In is any register. In this case, the address iscalculated as the sum of the current Program Counter value,the sign extended value in the register In and the signextended value of the eight bit label. The displacement inthis case is only 8 bits which yields a plus or minus 128byte range.

It is best to assume that the PCS option will generatesatisfactory position independent code when writing assemblylanguage code. The assembler cannot know of all violationsof the 32k byte reach of a displacement. This

knowledgebecomes evident at link time when link error messages willinform of any such violation. If such an error messageoccurs during the linking operation, a different approach isneeded to create position independent code for the branchesor instructions involved.

Position independent jumps can be created. Suppose we wish ajump operation that will move the program control to one ofseveral entries in a table based on a calculated value.Further, suppose that the range to these entries exceed thestandard plus or minus 32000 bytes that can be accessed by abranch operation. Any jump operation will require thedestination address. Therefore, the code involved mustcalculate the address of the destination. Examine thefollowing code sequence.

Enter this code sequence with the number of the procedure wewish to execute stored in ENT_NUMB. The table above containsthe relative addresses of the several procedures that can becalled by this sequence. The first instruction puts thenumber of the procedure into the data register D3. Eachentry in the table is four bytes long. The left shiftoperation multiplies the pointer value in D3 by 4 to point tothe correct location into the table.

Any BSR instruction pushes the value of the return addressonto the stack and transfers control of the program to thesubroutine address. In this case, we have placed thesubroutine as the next instruction following the BSR call.After the BSR instruction is executed, the address of thelabel HERE will be on top of the stack. The MOVE instructionat HERE removes this address from the stack and places it inA3. Regardless of the location of this code sequence inmemory, the address of HERE will be in A3 at this point.Adding the offset TABLE-HERE to the address will leave theaddress of TABLE in the address register A3.

The add ADD.L (A3,D3),A3 causes the value stored in the Nthtable entry to be added to the address of TABLE. The resultsof this calculation is the address of PROCN. The JSRinstruction then moves control to PROCN.

This fixed code sequence can be moved anywhere in memory andit will execute correctly. Note that the table entries willbe calculated at link time if the various procedures are indifferent modules. Also, it is not even necessary for thetable and the

```
        MOVE.L    ENT_NUMB,D3       Program number is in D3
        ASL.L     D3,2              Multiply pointer by 4
        BSR       HERE              Subroutine call puts
HERE    MOVE.L    (A7)+,A3          address of here in A3
        ADD.L     #TABLE-HERE,A3    Get address of TABLE
        ADD.L     (A3,D3),A3        Calculate the new PROCN
        JSR       (A4)              Go to it
.                                   Subroutine returns here
.
.
TABLE   DC.L      PROC0-TABLE       The relative addresses
        DC.L      PROC1-TABLE       of the several proc-
        DC.L      PROC2-TABLE       dures that can be
.                                   called.  These values
.                                   are calculated by the
.                                   linker.
.
```

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

MOTOROLA INC.

Dean Manley
Motorola
(512) 440-2839

Bill Bennett
Hewlett-Packard Company
(408) 447-0806

CONTACTS
Zachary Nelson
Cunningham Communication, Inc.
(408) 982-0400

HEWLETT-PACKARD DELIVERS LOW-COST 68030 WORKSTATION

68030 Offers Broad Range of Performance

AUSTIN, Texas, Dec. 5, 1 8 -- Hewlett-Packard Company announced that it will use the Motorola 68030 (030) microprocessor in a new low-cost engineering workstation. The HP 9000 Model 340, that also incorporates Motorola's 68882 (882) math coprocessor, is priced at $5,495.

Hewlett-Packard has more than 250,000 Motorola-based systems installed worldwide. Earlier this year, Hewlett-Packard added mid-range and high-end workstations based on the 030/882 combination to its 9000 Workstation Family, the HP Model 360 and HP Model 370. The Model 360 provides up to 5 MIPS (million instructions per second) of processing power and the Model 370 provides up to 8 MIPS. The new HP Model 340 complements these products by offering up to 4 MIPS of performance priced at $10,000 to $15,000 less than the 360 and 370.

Motorola's 68000 microprocessor line currently has four members — the 68000, 68010, 68020 and 68030. The development of the 68040 continues the evolution of the processor family. In total, 19 million chips from the 68000 family have been sold in applications including supercomputers, high-end workstations, business computers and embedded control devices. New generations of the 68000 are fully compatible with earlier 68000-based products; software written for one chip runs with no modification on the others, and hardware upgrades are very simple.

Hewlett-Packard is an international manufacturer of measurement and computation products and systems recognized for quality and support. The company's products and services are used in industry, business, engineering, science, medicine and education in more than 70 countries. Founded in 1939, the company celebrates its 50th anniversary in 1989. It has 87,000 employees and had revenue of $8.1 billion in its 1988 fiscal year.

Motorola's $2.2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a part of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America with a balanced portfolio of over 50,000 devices.

AA MOTOROLA INC.

EDITORIAL CONTACT
Dan Rogers, Motorola
512/441-6022
Or, Ken Phillips
602/952-3637

Semiconductor Products Sector
Public Relations
3102 North 56th Street
Phoenix, Arizona 85018
P.O. Box 52073
Phoenix, Arizona 85072-2073

FROM Canada C. Phillips
(602) 952-3637

A CHIP OFF THE NEW BLOCK

MESA, Ariz., Dec. 2 -- In the world of semiconductor manufacturing, Motorola's new MOS 6 high density CMOS wafer fabrication facility at Mesa, Ariz. is the industry's latest crown jewel, one whose products will help usher in a new era.

"This world class facility is Motorola's first U.S. manufacturing site to use some of the Toshiba process technology for manufacturing the latest architectures on six-inch wafers," said Marc Vandenberg, MOS 6 operations manager.

Vandenberg said the 38,000-square-foot, Class 10 factory is dedicated to fabricating one-megabit dynamic random access memories (DRAMs) and other memory circuits, as well as one of the semiconductor industry's hottest products, application-specific integrated circuits (ASICs).

The relationship with Toshiba, a Motorola technology partner and currently the world leader among one-megabit DRAM producers, also positions Motorola to use MOS 6 for future domestic manufacture of four-megabit DRAMs, which are expected to become the dominant memory chip by the early 1990s.

A one-megabit DRAM is capable of storing 1,048,576 bits of information, the equivalent of 64 double-spaced typed pages, on a single integrated circuit. The devices are used extensively in virtually all families of computers and computer-based systems, including computer-aided design, engineering, and manufacturing; communications; robotics; and consumer products.

ASICs, the second product type to be fabricated at the new Mesa facility, are fast becoming the semiconductor equivalent to the customized automobile. Designers select from a set of standard configurations, then add their own features depending on specific needs.

The result is reduced design and production costs and quicker turnaround, with completion of some types of semi-custom circuits in as little as two to three weeks.

Analysts estimate that ASICs, one of the fastest-growing segments of the semiconductor business, accounted for approximately 20 percent of the total market for integrated circuits last year.

Motorola managers said the advanced processes and quick cycle capabilities of MOS 6 are ideal for meeting rapidly changing requirements of high performance customer applications, especially in computers and communications.

The facility is a key element in Motorola's plans for the company's new High Density CMOS (HDC) Series, a line of ASIC products. Capable of processing sub-micron HDC parts as they evolve in the future, HOS 6 provides an immediate capability to manufacture the latest 1.0 micron HDC gate arrays.

Motorola's new ASIC circuits represent a significant advancement in microchip technology. Built with a triple layer metal (TRIM) CMOS process, they give semi-custom chip designers efficient routing and power distribution on a channelless architecture with minimal die sizes.

The result, according to Motorola managers, is a 75 percent gate utilization rate with very high performance (subnanosecond loaded gates) and unprecedented input/output flexibility and density.

"HOS 6 will provide advanced manufacturing technology administered by highly skilled people to support superior customer service and product quality, both in memory products and the rapidly expanding ASIC market," Vandenberg said.

88open

CONTACTS
James Strohecker
Cunningham Communication, Inc.
(408) 982-0400

Bob Anundson
88open Consortium, Ltd.
(503) 682-5703

88open Consortium Holds Member Meeting Completes Binary Compatibility Standard

Two New 88000-based Systems Displayed At The Meetings

WILSONVILLE, Ore. — December 12, 1988 — The 88open Consortium, Ltd. today announced the results of its members meeting held last week, where 128 representatives from more than 50 companies rallied to support the Consortium's standards development and "Team Computing" concept. At the meeting, two hardware vendors displayed yet-to-be-announced computer systems based on Motorola's 88000 RISC microprocessor.

During the two day meeting, held in San Diego, Calif., the 88open Binary Compatibility Standard (BCS) committee met and approved the final BCS that allows 88000 RISC-based application software to operate transparently across numerous systems, similar to PC "shrink wrapped" software.

More than 20 international representatives from European companies met separately at the meeting to review overseas standards development, product announcements and the appointment of an 88open European director. Many of the international representatives indicated that their companies would announce 88000-based hardware and software developments in early to mid 1989.

The meeting provided new and existing 88open members with the opportunity to meet and review progress of software application, hardware and standards development. The members discussed updates on operating system based groups such as the Open Software Foundation (OSF) and Unix International (formerly the Archer Group), upcoming product announcements and individual and collective publicity efforts.

Chuck Corley, manager of Motorola's 88000 Microprocessor Development, presented a technical and architecture update, including the general sampling schedule and the announcement of improved clock speeds with upcoming versions of the 88000 architecture.

"It was impressive to see that in just six months we evolved from a group of 30 vendors who were reluctant to let each other know which company they represented, to a cohesive representation of more than 125 people who were working together to develop standards and bring about the success of the 88000 architecture," said Bob Anundson, executive director of the 88open Consortium. "We believe that this form of Team Computing is what other RISC vendors must ultimately turn to for true success across the RISC architecture and hardware and software."

The Binary Compatibility Standard is a cornerstone for making the Team Computing environment possible. 88open's concept of Team Computing enables all application software that complies with the BCS to run on any of the 88000-based hardware systems. User benefits include a highly competitive environment with a vast array of cost-effective solutions.

The BCS has been under development by a committee of 88open members since the introduction of the Motorola 88000 RISC architecture in April and is the first RISC standard to be made available in the industry. The 88open BCS is based on AT&T's Unix System V Rel. 3.0 and combines X/Open and IEEE standards. Interested parties may obtain the BCS from 88open.

A number of new members including NCR were added to the 88open roster at the meeting. It was announced that Richard Herter of NCR was appointed a seat on the 88open board of directors. NCR has secretly been a Consortium member since the formal introduction of the 88open in April 1988.

The 88open Consortium, Ltd. is a not-for-profit organization formed to develop and promote the success of Motorola's 88000 RISC microprocessor architecture. The Consortium, based in Wilsonville, Ore., includes more than 51 worldwide members. 88open is pioneering a unique way of doing business built on the portability of 88000-based applications across a variety of hardware platforms including fault tolerant systems, computer servers and workstations. Information regarding the Consortium can be obtained from the 88open Consortium, Ltd. at 8560 SW Salish Lane, Suite 500, Wilsonville, OR 97070; telephone (503) 682-5703.

SANYO/ICON FIRST 88OPEN MEMBER TO DISCLOSE 88000-BASED MULTI-USER SYSTEM

WILSONVILLE, Ore. — December 12, 1988 — The 88open Consortium, Ltd. today announced that Sanyo/Icon International (Orem, Utah), an American subsidiary of the Sanyo Electric Co. (Osaka, Japan), is the first 88open member to introduce a Motorola 88000 RISC processor-based system. The system, called Icon 8000, is a multiuser business system which supports up to 256 users, delivers 15 MIPS (millions instructions per second) of performance and is the first of a number of products from a variety of vendors to comply with the 88000 Binary Compatibility Standard (BCS). The BCS allows 88000 RISC-based application software to transparently operate across numerous systems, similar to PC "shrink wrapped" software.

Sanyo/Icon is one of more than 45 member companies of the 88open. The company joined the 88open Consortium in June of 1988. The 88000 architecture, which includes many features of the so-called "superminicomputer" market, is currently being evaluated by more than 200 manufacturers worldwide.

"The Sanyo/Icon announcement demonstrates the first implementation under a new computing environment we call 'Team Computing,'" said Bob Anundson, executive director of the 88open Consortium. "Team Computing is where all application software will run on any of the 88000-based hardware systems. The benefits to the user are a highly competitive environment with a vast array of cost-effective solutions."

The BCS is a cornerstone for making the Team Computing environment possible. That standard has been under development by a committee of 88open members since the introduction of the 88000 chip set in April, and was the first to be made available in the industry. The 88open BCS is based on AT&T's UNIX System V Rel. 3.0 and combines X/Open and IEEE standards. Interested parties may obtain the BCS from 88open.

The 88open Consortium, Ltd. is a not-for-profit organization formed to develop and promote the success of Motorola's 88000 RISC microprocessor architecture. The Consortium, based in Wilsonville, Ore., includes more than 45 worldwide members. 88open is pioneering a unique way of doing business built on the portability 88000-based applications across a variety of hardware platforms including fault tolerant systems, computers servers and workstation. Information regarding the Consortium can be obtained from the 88open Consortium, Ltd. at 8560 SW Salish Lane, Suite 500, Wilsonville, OR 97070; telephone (503) 682-5703.

EKF-ELEKTRONIK-MESSTECHNIK GMBH
Systemhaus für Microcomputer und Industrie-Elektronik

ekf-system

Wiesbadenstraße 1a, D-4700 Hamm 1
Telefon (02381) 12630
Telex 828621 ekf d
Telefax (02381) 15067

Don Williams
68 Micro Journal
Computer Publishing Center
5900 Cassandra Smith Road

Data-Path Switchbox for V.24/RS232/RS423 * 68520-SX

Two independant sections of 3-way data communication path selectors for serial interfaces according EIA RS232 are contained in the 68520-SX switchbox from EKF, thus allowing coupling of several peripheral devices to more than one computer system. Integrated null-modems are provided to simplify changing of DTE to DCE and vice versa, so that connections from computer to computer or terminal to terminal require no more special cabling hardware.

Housed in a steel-sheet box for electrical and magnetical screening, 10 DB-25 female connectors are routed on the pin's Tx-Data, Rx-Data, CTS und DTR (No. 1, 2, 3, 5, 7, 20). Frame (Pin 1) is available via 4mm grovading socket.

Bankverbindungen:
Postscheck Dortmund (440 100 46) 129811-467
Sparkasse Stadt Hamm (410 500 95) 3 016 367
Deutsche Bank Hamm (410 700 48) 0 327 254

ekf-system

Handelsregister Nr. B 890, Amtsgericht Hamm

EKF-ELEKTRONIK-MESSTECHNIK GMBH
Systemhaus für Microcomputer und Industrie-Elektronik

ekf-system

Wiesbadenstraße 1a, D-4700 Hamm 1
Telefon (02381) 12630
Telex 828621 ekf d
Telefax (02381) 15067

Don Williams
68 Micro Journal
Computer Publishing Center
5900 Cassandra Smith Road

EKF Adapterboard changes 68020 systems to 68030 CPU * VME 68061-ADP30

With dimensions of only 38x76mm, the EKF adapterboard VME 68061-ADP30 allows upgrading of any existing 68020 microcomputer system to the more advanced 68030 chip. The only thing the user has to do is to remove the 68020 CPU and to insert the adapter, carrying the 68030 microprocessor.

The pc-board is a 6-layer construction with power and groundplane in combination with additional decoupling capacitors for effective noise reduction to insure absolutely reliable operation. Existing 68020 software runs on the 68030 microprocessor without any changes, such that upgraded computers can be booted from harddisk or diskette in the same manner as before. The VME 68061-ADP30 adapter is available from EKF stock at a single quantity price of DM 220.- (Europe) or US$ 135.- (USA).

Bankverbindungen:
Postscheck Dortmund (440 100 46) 129811-467
Sparkasse Stadt Hamm (410 500 95) 3 016 367
Deutsche Bank Hamm (410 700 48) 0 327 254

ekf-system

Gespac, Inc

50 West Hoover Ave.
Mesa, Arizona 85210
Tel. (602) 962-5559
Fax. (602) 962-5750

Reader Contact: Don Bizios
Editorial Contact: Cosma Pebouctsidis

GESPAC ISSUES 1989 CATALOG

MESA, AZ, December 9, 1988--GESPAC, Inc has released its new 1989 product catalog of board level products, software and systems. The 116 page, all-color document is the largest the company has ever produced and lists over 300 up-to-date product references.

The GESPAC catalog is the largest in the board level industry and includes the most diversified variety of hardware and software components. The GESPAC catalog offers as much, if not more choice, than the typical bus buyer's guides of other architectures. The GESPAC catalog is sent free of charge to qualified engineers.

The catalog is divided into several Major product categories to help guide the engineer's selection. The principal categories are: Intel and Motorola based Processors; static and dynamic Memories; serial, parallel and analog Interfaces; Controllers for disks, graphics, data communication, networking and motor control; various accessories such as card cages, cables and backplanes.

A new product category lists a complete offering of linear scan cameras and controllers.

The catalog's software section has grown to include a library of software drivers for most boards offered by GESPAC, software development utilities, in addition to the usual real-time operating systems and high level languages.

An important section of the catalog is dedicated to systems. This section lists a large family of pre-assembled and ready-to-use development systems and OEM computers based on the boards and software developed by the company. These systems are either based on the UNIX-like OS-9 real-time operating system or on the popular MS-DOS operating system.

The catalog also lists Application Specific Integrated Circuits developed by GESPAC for interfacing to the G-64 bus. These ASICs are made available to all third party board manufacturers.

FORCE COMPUTERS INC
3165 Winchester Blvd
Campbell, CA 95008-6557

Product News

Prepared By:
Shohat & Kahn PR
2959 S. Winchester Blvd, Campbell, CA 95008
Murry Shohat (408) 379-7434

Contacts:
FORCE USA, Wayne Fischer (408) 370-6300
FORCE GmbH, Anton Nausch (089) 600-910

First 68030-based VMEbus Message Passing CPU For Distributed Real-Time Multiprocessing at the High End

CPU-30 Selected For Navy Simulator Program, Displacing Minicomputers & Mainframes

CAMPBELL, CA., December 13, 1988 — The highest known performance 68030-based VMEbus single board computer began shipping this month. FORCE COMPUTERS believes the CPU-30 sets new marks in functionality and processing throughput that will expand the use of VMEbus products into new markets. The CPU-30 is the first true message passing single-board computer available for VMEbus applications. Its introduction begins an era of off-the-shelf solutions for applications previously served by more expensive minicomputers and mainframes.

The CPU-30 has already been selected for use in a major Navy simulator program (see separate release) that stresses these features.

"The CPU-30 enables the design of very large scale high performance systems in which the processing power is distributed among users rather than concentrated in a centralized computer," said Manny Weisberg, FORCE COMPUTERS Vice President & General Manager. "No other VMEbus solution offers the message exchange architecture needed for real-time distributed multiprocessing. The result is much higher performance at lower cost," he added.

Litany of Features Makes CPU-30 More Like Minicomputer Than VME CPU

The CPU-30 exceeds the functionality of any known VMEbus board to achieve the status of a true system-on-a-board. Its main features include:

- 68030 microprocessor, 20 or 25MHz operation
- 68882 floating point coprocessor, 20 or 25MHz
- High performance 32-bit DMA controller – high speed data transfers locally and across the VMEbus; 32 byte internal FIFO for burst DMA
- VME/PLUS™ technology implemented in FORCE GATE ARRAY 2 (FGA-002), a 22,000-gate ASIC that provides message passing, mailbox interrupts and a comprehensive VMEbus interface
- 4 Mbytes system memory; shared dynamic RAM with byte parity; accessible from the VMEbus via FGA-002
- Message broadcast to up to 20 CPUs simultaneously
- SCSI interface using on-board DMA controller
- Floppy disk interface (SA450 compatible)
- Four multiprotocol serial ports, 1 channel is RS232 and 3 are RS232/RS422/RS485 compatible
- 8-bit parallel interface with handshake
- Up to 4 Mbytes of EPROM with support for 28- and 32-pin devices, 32-bit data path
- System EPROM for local boot and smart initialization of the I/O interface and FGA-002
- Real-time clock/calendar with battery back-up
- 32 Kbytes of local high-speed SRAM with battery back-up
- Optional 10 Mbit/second Ethernet controller
- 2 24-bit and 1 8-bit timers
- Full 32-bit VMEbus master/slave interface, single-level arbiter, SYSCLK driver, interrupt handler, support for ACFAIL & SYSFAIL
- VMEPROM™, FORCE COMPUTERS' standard real-time operating system kernel

Multiprocessing Capability & VME Functions are Second to None

The "crown jewel" of the CPU-30's architecture is FGA-002, a 281 pin ASIC that enables multiprocessing via message passing and mailbox interrupts. FGA-002 also provides all the glue logic needed by the host processor to drive the VMEbus and perform the industry's most comprehensive set of master, slave and control functions. These features are collectively labeled VME/PLUS. On-board performance is boosted because the ASIC minimizes gate delays.

As a multiprocessing engine, the CPU-30 implements message passing modeled on minicomputer and mainframe ideals, but with VMEbus compatibility. Message passing permits any processor to broadcast, at any time, status, data, interrupt or other messages to a select but potentially large set of receivers. Because message passing enables real time system operation, a backplane populated with several VME/PLUS CPU's can handle a vast array of functions without contention or system crashes.

The CPU-30 employs an 8-bit wide, 8-byte deep FIFO message buffer and an 8-bit wide high priority message register. The byte-wide configuration enables a scheme of 256 possible messages. Any VME master can deposit a message in the buffer or register. Up to 20 VME/PLUS—equipped boards can simultaneously receive a message. Message receipt stimulates quick action, typically on a predetermined basis. For example, servicing the interrupt caused by message receipt may turn-off an alarm, move a robot arm, calculate a block of data or send another message to another CPU, etc. The action taken is totally software programmable. It is easily tailored to specific applications. And, it happens more swiftly and with greater certainty of success than any other architectural approach that complies with the IEEE-1014 industry-standard VMEbus specification.

Directed Software Interrupts via Mailboxes

The VMEbus provides 7 hardware interrupt levels; this restricts the number of active interrupt handlers to 7 at a time. The CPU-30 augments this by providing 8 Mailbox interrupts which are software driven. With 8 per board, a fully populated backplane (21 CPUs) provides 168 directed software interrupts for handling the interrupt volume expected in complex systems. The use of Mailboxes allows swift execution through quick generation of interrupts, and results in incredible flexibility.

Corporate News

Prepared By:
Shohat & Kahn PR
2959 S. Winchester Blvd, Campbell, CA 95008
Murry Shohat (408) 379-7434

Contacts:
FORCE USA, Wayne Fischer (408) 370-6300
CAE-LINK, Bob Taggart (301) 622-4400

FORCE VME/PLUS™ Technology Chosen by CAE-LTSD for Navy Simulator

Largest Contract in FORCE History Also Marks Penetration of VMEbus into Minicomputer and Mainframe Territory

ORLANDO, FL., November 30, 1988 — Link Tactical Simulation Division (LTSD), located in Silver Spring, MD, (a subsidiary of CAE Industries located in Toronto, Ontario, Canada) and FORCE COMPUTERS (Campbell, CA) have announced award of a major subcontract for computer subsystems for the U.S. Navy's 14A12 Surface ASW (Anti-Submarine Warfare) Trainer.

CAE-Link is the prime contractor for the production 14A12. The subcontract with FORCE will result in the supply of very high performance VMEbus-based computers that provide both student and instructor stations in an elaborate, extremely realistic re-creation of surface ASW strategies and tactics.

Under terms of the contract, FORCE is implementing the CAE-Link design and delivering fully integrated, functional subsystems.

The announcement was made on the eve of the annual I/ITSC Conference (Interservice/Industry Training Service Conference), held here. At its exhibit booth, CAE-Link will demonstrate working student stations developed with FORCE during a development phase. FORCE also plans to demonstrate working models of the VMEbus computer design used in the program. It will be the first public showing of these extremely powerful computers.

The 14A12 ASW Trainer is an aggressive Navy program to build an ASW training capability for the crews of surface ships. The program has an expected procurement life of several years, during which the FORCE subcontract could be worth more than $10 million. Delivery of the FORCE hardware begins in early 1989.

FORCE Chosen For Product Performance, Design Synergy

Bob Taggart, CAE-Link's Vice President of Marketing, said "the FORCE VMEbus-based solution is a sharp departure from the minicomputers that usually get the nod for this type of work. VMEbus hardware will change the face of simulation because it's inherently more powerful, easy to distribute where needed, consumes much less power and has much lower life cycle cost. This approach results in modular building blocks that can be used in other trainers. We chose FORCE because their latest VME technology offers much higher performance and the ability to distribute processing power and multiprocessing capability in a complex simulation involving many students and instructors. Their real-time hardware is not only more powerful, it's also less expensive."

Taggart also indicated that FORCE was chosen over several competitors "because they joined our design team and provided the ongoing support and synergy that helped us to win the Navy contract."

Manny Weisberg, FORCE Executive Vice President and General Manager, said "CAE-Link's leadership in simulation continues because they were willing to try something new. FORCE is using its latest design for the 14A12. These boards employ the 68030 microprocessor in an exciting new architecture called VME/PLUS that provides mainframe computer qualities such as message passing and mailbox interrupts. We think this contract marks an era in simulation and training procurement using off-the-shelf industry-standard solutions."

During the year-long development phase of the 14A12, FORCE provided hardware and also prepared a real-time operating system, pSOS.

Trainers Will Enhance Readiness

The 14A12 Surface ASW Trainer has the potential to improve combat readiness. The Navy's plan call for construction of several 14A12 Trainers. Weisberg said the use of VME/PLUS distributed multiprocessing in the 14A12 design was being looked at closely as a candidate for other simulation and training programs across the military services and commercial airlines.

About Force Computers

The leading independent designer and manufacturer of VMEbus products, Force is entering its seventh year. The company has completed 23 consecutive quarters of profitable operation. Force is headquartered in Campbell, California with subsidiaries in West Germany, France and the United Kingdom. Sales, service and product support are provided on a worldwide basis.

About CAE-Link

Link Tactical Simulation Division is the leading producer of training simulators for the U.S. Navy's surface, air and subsurface ASW applications; surface team tactics training and ground C^2 systems for the U.S. Army.

MICRONICS RESEARCH CORP.

Microcomputers - Hardware and Software
GIMIX® Sales, Service and Support

33383 LYNN AVENUE,
ABBOTSFORD,
BRITISH COLUMBIA,
CANADA. V2S 1E2

Dear Don,

A preview of friend Denis' 68U3 for December shows a few typos, which readers may wish to correct. These are :

P16, last para should read "What does S_1^3 ABC' convey"
Diagram 85, row 3, col 10 should read 120, not 126.
P20, last para but one, should read "If they're equal, we get directed to BLOCK-4a ..."
P22, para 6, should read "... function as S_{12}^3 ABC' .."

I've also noticed a sneaky one in the August issue, page 21, near top, should read n2 n1 n2 n1. The n2/n1 combo somehow got reversed!

I've been asked by a few readers about Manuals for RBASIC. The price for the manual alone is US$25, including postage for North America (plus $5 elsewhere), $22 of which will be applied against a future purchase of RBASIC.

Don Williams,
68 Micro Journal,
5900 Cassandra Smith Road,
Hixson, TN 37343

Sincerely,

R. Jones
President

B&R Industrial Automation Corp. is a manufacturer of programmable logic controllers, industrial computers, visualization and process control systems.
B&R offers the own hardware and software with easy communication to other systems.

One of the latest developments is a OS9 based minicomputer system with floppy controller on board, external harddisk (40 MB), optional memory extension board (RAM and EPROM) and a high resolution graphic controller. This so-called MAESTRO-System is available for OS9 based software or can be used with B&R standard software and offers direct access to digital and analog inputs and outputs.

For more information please contact: B&R Industrial Automation Corp.
2165 West Park Court
Stone Mountain, GEORGIA 30087
Tel: 404 469 4617, Fax: 404 469 5460

H.C. ANDERSEN COMPUTER A/S
Th. Philipsensvej 21-23

DK-2770 Kastrup, Denmark
Telephone: 45 1 52 44 04
Telex: 31484 tmrcph

License: OS9/6809 Level 1 (DRAGON Computer)
Development: OS9/68000 ATARI ST (Dr.Keil)
OS9/68000 ATARI ST (Cumana)
Drivers for t2->t5, cadscrn /term with special fonts, TOS GATEWAY (PD)
SCULPTOR development

OS-9 for VME: Single Boards to Complete Systems

Mizar provides complete OS-9 solutions for the VMEbus. Mizar's VME CPUs offer the functions and performance your application demands. Our single-height (3U) VME processors are uniquely configurable computing engines. Through Mizar's unique MXbus™ expansion interface, standard and custom side modules can be added to basic processors to create double-height (6U) boards for specific applications. 3U CPU options include 68010, 68020, and 68030 microprocessors, up to one MB of DRAM, serial I/O, real-time clock, and mailbox interrupt support. Standard MXbus side modules include additional DRAM, SRAM, and I/O.

Mizar's standard double-height (6U) processors provide additional features such as a high-speed cache to enhance 68030 performance, floating point coprocessor support, up to four MB dual-ported DRAM, VSB memory interface, Ethernet, and SCSI.

Mizar also supports OS-9 with completely configured OS-9 development systems and OS-9 application server systems.

For more information, call Mizar today.

800-635-0200

MIZAR

1419 Dunn Drive • Carrollton, TX 75006 • 214-446-2664

Pan Controls Limited specifically provides services in the field of industrial automation. OS9 is used for the majority of the high level computing work, and a number of software packages have been developed for sale generally.
STIMULUS is a general purpose rule-based expert system programming language designed for easier and more economic real-time processing for engineering control, simulation, monitoring, diagnosis, etc. Since STIMULUS produces C source, it may be linked with other modules in C or Assembly language, giving you the choice of C's I/O and interrupt facilities.
PROFILE is a symbol profiling tool for C or Assembly language programs that increases programmer efficiency. It provides a valuable and quick method of optimising large and small projects by making a table of the time spent running each of your functions in your programme during a working session. A quick look at this profile table shows you which functions are most used and help you to concentrate your programming efforts on optimising these functions.
PAN UTILITIES is a library of programs providing the major text-handling, file-handling and programmers utilities which the OS9 operating system lacks.

PEP Modular Computers is one of the leading manufacturers of microcomputer boards and systems for the VMEbus and IIOC (Intelligent I/O Channel) in compact single height format./Founded in 1975 in Kaufbeuren, Germany, PEP is now established worldwith with subsidiaries in USA, France an Sweden./Today PEP offers a broad range of VME and IIOC board level products, plus development systems, racks, backplanes, power supplies and standard operating systems./ PEP's target market segments are industrial control applications including machine control, process control, data acquisition and robotics. This year PEP introduced for example VSCSI, a SCSI interface module for the VMEbus, VDAO, a combined A/D, D/A and digital I/O VME single height module, VMPM68KC-2, a powerful 68020 VMEbus CPU in military temperature range, VLAN, a cost-effective VME networking solution, VGPM a high performance graphics module based on the Hitachi ACRTC63484. The company also introduced VIOXROM, an universal VME software interface for a family of intelligent VME I/O modules./ Currently about 100 people are working for PEP worldwide. For further information please contact: Germany 08341/81001

Scorpion Technologies Inc. features a line of Motorola 680X0 based co-processor add-in boards for IBM PC/XT/AT and compatibles providing the most economical development platform for OS-9 engineers and programmers. This is the only known implementation of Professional OS-9 available on the PC platform and provides the unique feature of PC DOS concurrency. The Pro68 family is available with choice of 68000, 68010 or 68020 processors and a minimum of 1MB of memory (expandable to 4MB) Two serial ports are available on the 68010 and 68020 versions with support for 16 users via intelligent serial I/O cards.

UNIVERSAL ETHERNET CONTROLLER BOARD
MPVME1054
* Intelligent Ethernet Slave Board running in any VME System

A TRUE MICROCOMPUTER BOARD SOLUTION WITH ETHERNET CAPABILITY
MPVME 1002 (16 Bit) MPVME1021 (32 Bit)
* Three communication packages under OS-9/68K
for all three Board available:
- DECNET/VME
- TCP/IP
- OS9-NET

Call for a complete documentation

SYSTEMFORSCHUNG, Königstr. 33a, 5300 Bonn 1, W-Germany Tel. 0228/223151

XYZ Electronics, Inc. RR 12, Box 322 Indianapolis, IN 46236
(317) 335-2128 Contact: Gary Bannister
10-year old company manufacturers STD bus based industrial computer systems and boards/ Suport OS-9/6809 Level I and Os-9/68000 Professional Package.
Board level products include CPU-9A 2MHz 6809, and CPU-68K3 10 MHz 68009. Both boards support STD bus for memory and I/O expansion.
Board level products, board sets, and complete systems available.
Memory, Floppy and hard disk controllers, analog I/O, digital I/O, and cages available.

STD Bus systems offer a superior price/performance ration where medium level performance in an industrial environment is required.

ATARI & AMIGA CALL

As most of you know, we are very sensitive to your wishes, as concerns the contents of these pages. One of the things that many of you have repeatedly written or called about is coverage forthe **Atari & Amiga™** series of 68000 computers.

Actually we haven't been too keen on those systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari & Amiga machines. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari & Amiga computers. Especially so since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you) and rumored for the Amiga. Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari & Amiga.* As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. Which is the way it has been for over 10 years. We need articles, technical, reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari & Amiga systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it, drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly LanRuaRe Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled **Operating Programs.** The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source ListinRs
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leadinR microprocesaor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks.

LOGO.C1	File load program to offset memory — ASM PIC
MEMOVE.C1	Memory move program — ASM PIC
DUMP.C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST.C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM.C2	Modem input to disk (or other port input to disk) — ASM
M.C2	Output a file to modem (or another port) — ASM
PRINT.C3	Parallel (enhanced) printer driver — ASM
MODEMC2	TTL output to CRT and modem (or other port) — ASM
SCIPKG.C1	Scientific math routines — PASCAL
U.C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT.C4	Parallel printer driver, without PFLAG — ASM
SET.C5	Set printer modes — ASM
SETBAS1.C5	Set printer modes — A-BASIC

NOTE: .C1,.C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included in ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
*** All Currency in U.S. Dollars**

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

(615) 842-4601
Telex 5106006630

"FLEX is a trademark of Technical Systems Consultants
"OS9 is a trademark of Microware and Motorola
"68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard ☐ VISA ☐

Card #_____ Exp. Date_____

For 1 Year ____ 2 Years ____ 3 Yeais ____

Enclosed: $_____

Name _____

Street _____

City _____ **State** ____ **Zip** _____

Country _____

My Computer Is: _____

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

***Foreign Surface: Add $12.00 per Year to USA Price.**

***Foreign Airmail: Add $48.00 per Year to USA Price.**

***Canada & Mexico: Add $9.50 per Year to USA Price.**

***U.S. Currency Cash or Check Drawn on a USA Bank !**

68 Micro Journal
5900 Cassa dra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry, **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Prmod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Hatkness.
Disk- 8	Crtest, Lanpher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign,Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong),CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchrist). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven. Language Recognition Utility, Anderson March 1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.
Disk-34	FLEX 6809 Diagnostics, Disk Drive Test, ROM Test, RAM Test - 68MJ 4/88Koipi.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9 FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:
~~$199.00~~

- 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

- 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

- 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program In BASIC, Debug it In BASIC and Then Compile It to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:
$69.95

SPECIAL
Thank-You-Sale

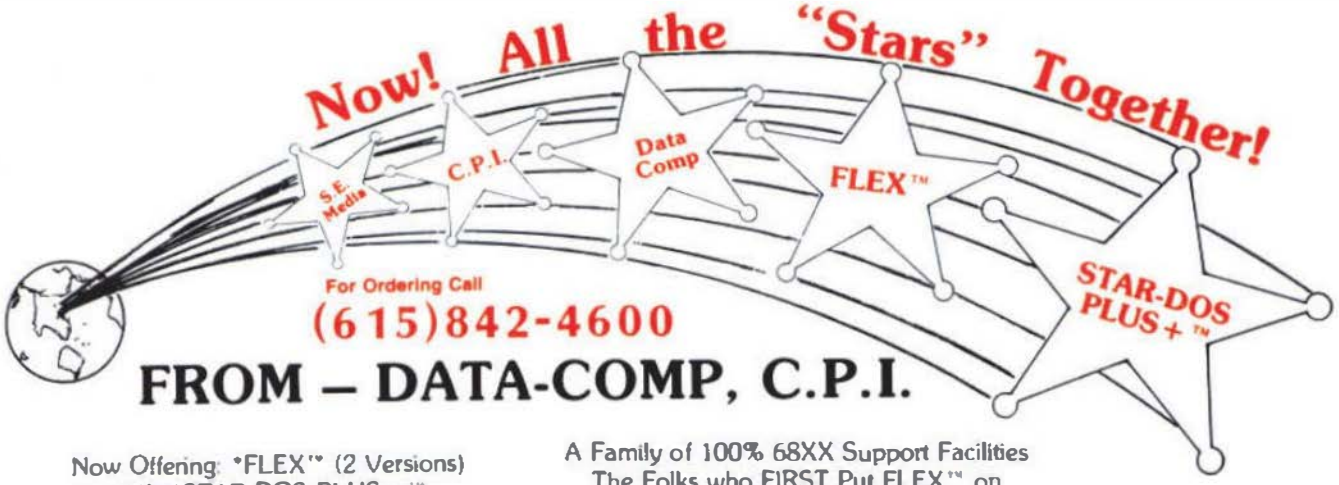
Only From:

CPI **S.E. Media™**
5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.

Price subject to change without notice.



FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

$34.95

FLEX-CoCo Jr.
without TSC Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Editor
Reg $50.00
NOW $35.00

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Systems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITH POWER SUPPLY,CABINET,DISK DRIVE CABLE,J&M NEW DISK CONTROLLER JFD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95
* Specify What CONTROLLER You Want J&M, or RADIO SHACK	
THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JFD-CP WITH J-DOS	$139.95
WITH J-DOS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,P, AND COCO 11	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$325.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95
CABLES & OTHER INTERFACES CALL FOR PRICING	

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

MasterCard VISA

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

(615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The MUSTANG-08™ system took every hand from all other 68008 systems we tested, running OS-9 68K!

The MUSTANG-08 includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actualy a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed *whiz on disk I/O.* Fact is the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks!

SALE!!

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K No Wait States	256K Chips
PORTS	4 - RS232 2 - 8 bit Parallel	MC68681 DUART MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

MUSTANG-08

LOOK

					Seconds	32 bit Integer	Register Long
Other	68008	8 Mhz	OS-9	68K	18.0	9.0	
MUSTANG-08	10 Mhz		OS-9	68K	9.8	6.3	

```
Main()
{
                    /* int i; */
C Benchmark Loop    register long i;
                    for (i=0; i < 999999; ++i);
}
```

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk	
MUSTANG-08 8 Mhz CPU	0 min - 32 sec
Other popular 68008 system	1 min - 05 sec
MUSTANG-020	0 min - 21 sec

25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC MUSTANG-08 is a trademark of CPI

Data-Comp Division

CPI — A Decade of Quality Service* — Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road

Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC Hi-density FLEX 5" - Call for special info.